uOttawa

L'Université canadienne
Canada's university

CEG2536 Architecture des ordinateurs I

Organisation de base d'un ordinateur

Dr. Mohamed Ali Ibrahim

SITE: local 0110B

mibrahi3@uottawa.ca

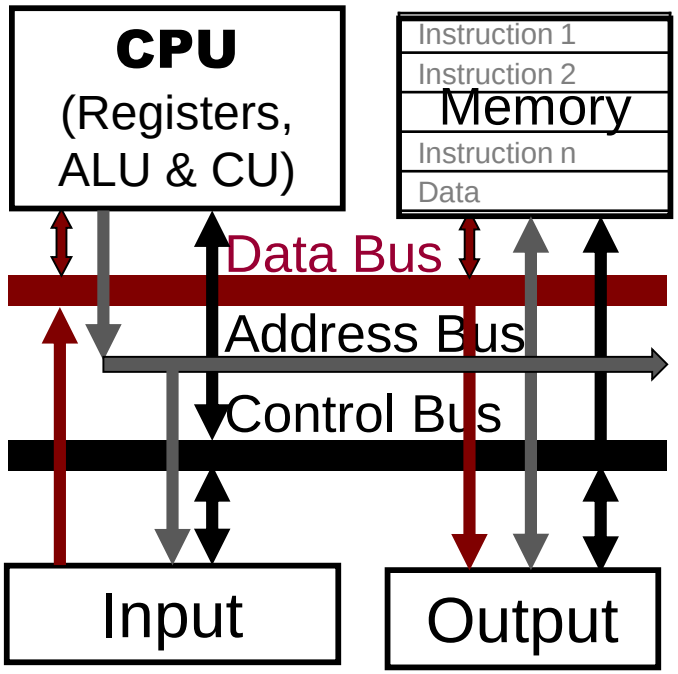
Université d'Ottawa | University of Ottawa

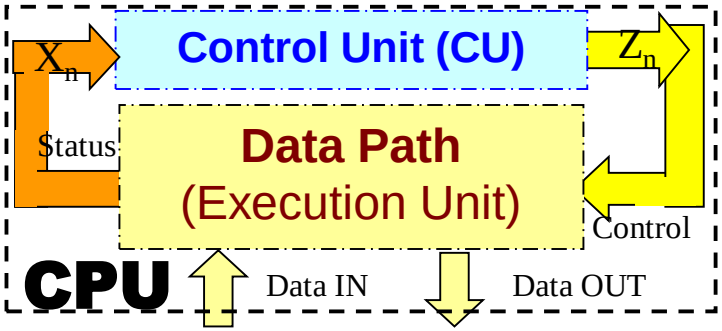
www.uOttawa.ca

Plan

- Spécification du problème:
  - Codes d'instruction de base de l'ordinateur
  - Modes d'adressage
  - Le chemin de données de base de l'ordinateur
  - Instructions de base de l'ordinateur
    - Registre - Instructions de référence
    - Instructions de référence de mémoire
    - Entrée-sortie et interruption
  - Configuration requise pour l'unité de contrôle (Control Unit, CU) de base d'ordinateur
- ASM = outil (méthodologie) pour résoudre le problème
- Conception de la CU de base de l'ordinateur

Fonctionnement de base d'un ordinateur

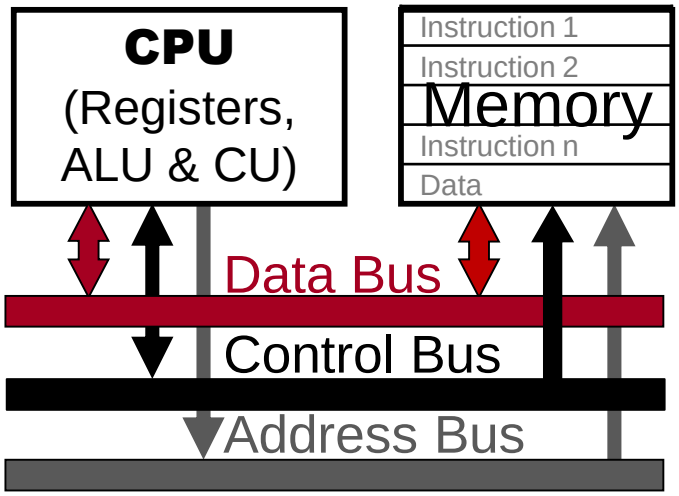




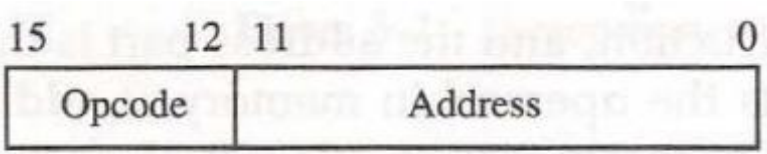
1. L'ordinateur accepte des informations externes sous forme de séquence d'instructions (programmes) et de données via une unité d'entrée et les stocke en mémoire.
2. Les données stockées en mémoire sont récupérées, contrôlées par le programme et traitées dans une unité ALU.
3. Les données traitées quittent l'ordinateur via une unité de sortie
4. Toutes les activités à l'intérieur de l'ordinateur sont dirigées par l'unité de contrôle (CU).

L'unité de contrôle (CU) du CPU est un circuit séquentiel «programmable» doté de fonctions de transition différentes pour chaque instruction.

Code d'instructions



- Un code d'instruction est un groupe de bits (code binaire) qui ordonne à l'ordinateur d'effectuer une séquence de micro-opérations.
- Les codes d'instructions ainsi que les données (opérandes) sont stockés dans la mémoire de l'ordinateur.
- L'unité de contrôle (CU) interprète ensuite le code binaire de l'instruction et l'exécute en effectuant une séquence de micro-opérations.
- Un code d'instruction est généralement divisé en deux parties: un code d'opération (opcode) et un code d'adresse.
- L'opcode (code d'opération) définit l'opération à effectuer: addition, soustraction, ET logique, etc.
- Le code d'adresse spécifie l'adresse de l'opérande pouvant être en mémoire, pour les instructions de référence de la mémoire (Memory Reference Instructions, MRI) ou en
  - mémoire.
  - registres, pour les instructions de référence du registre (Register Reference Instructions, RRI) ou instructions de référence d'E/S (IOI).



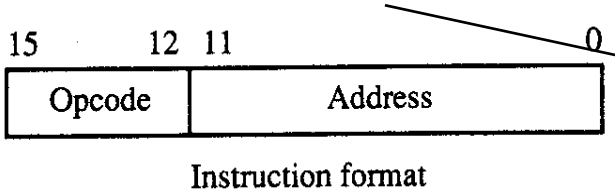
Instruction format

Code d'instruction

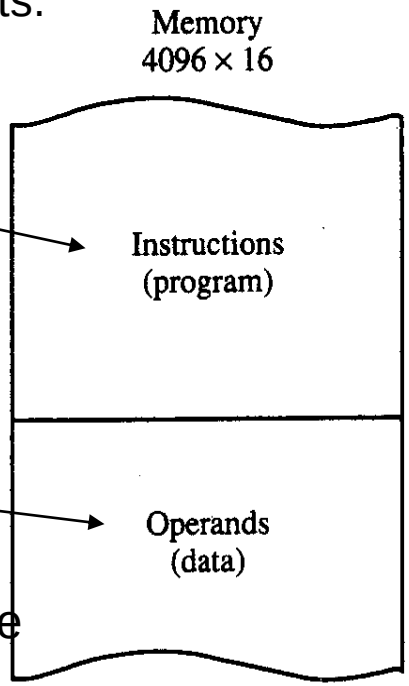
- L'unité de commande reçoit le code d'instruction de la mémoire, interprète le code d'opération (opcode), puis émet une séquence de signaux de commande pour initier la séquence nécessaire de micro-opérations à exécuter sur les opérandes spécifiées.
- Un code d'instruction doit spécifier non seulement l'opération (définie par le code d'opération), mais également les registres ou les emplacements de mémoire où trouver les opérandes, ainsi que l'emplacement de registre ou de mémoire où le résultat doit être stocké.
- Un emplacement de mémoire peut être spécifié dans le code d'instruction en spécifiant son adresse en mémoire.
- Un emplacement de mémoire peut être spécifié dans le code d'instruction en spécifiant son adresse en mémoire.
- Un registre de processeur peut être spécifié dans le code d'instruction en affectant k bits qui identifient l'un des 2^k registres de processeur possibles disponibles dans le système.
- Dans ce chapitre, nous aborderons l'organisation de base, les fonctionnalités et la conception d'un système informatique à petite échelle.

Organisation du programme stocké

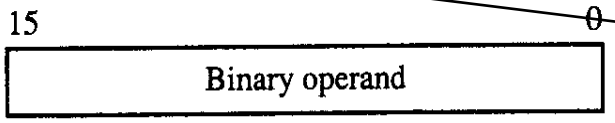
- Une unité de mémoire 4096 × 16 est utilisée ici ⇒ mots 4096 × 16 bits.
- La mémoire est divisée en deux parties:
  - Une partie de programme, stockant le programme (ensemble d'instructions) à exécuter,



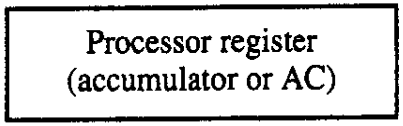
Instruction format



  - Une partie de données stockant les données (opérandes) à utiliser pour les instructions du programme.

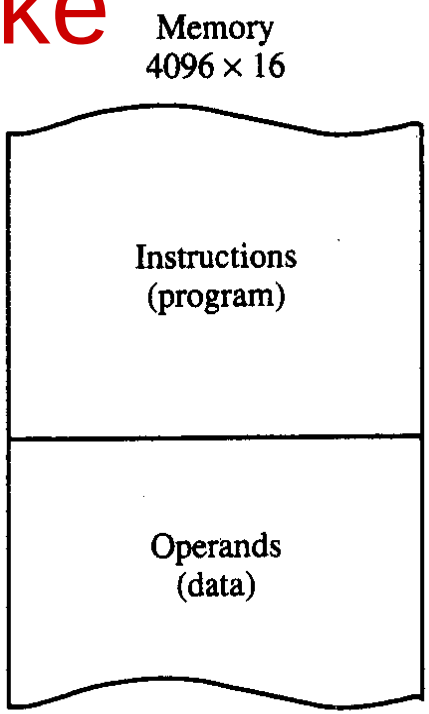


- Chaque instruction dans la partie programme de la mémoire a une longueur de 1 mot (16 bits) et est divisée en deux parties:
  - La première partie est composée de 4 bits (bits 12 à 15) qui spécifient le code d'opération (**opcode**) de l'une des $2^4 = 16$ opérations possibles au maximum.
  - La seconde partie est composée de 12 bits (bits 0 à 11). Elle spécifie l'adresse de l'opérande en mémoire (MRI) ou peut coder d'autres opérations ne nécessitant pas d'accès à la mémoire (RRI ou IOI).

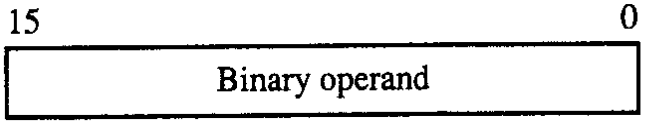


- Les 12 bits de la partie adresse du code d'instruction peuvent spécifier au maximum $2^{12} = 4096$ adresses de mots différentes dans la mémoire.

Opération de programme stocké

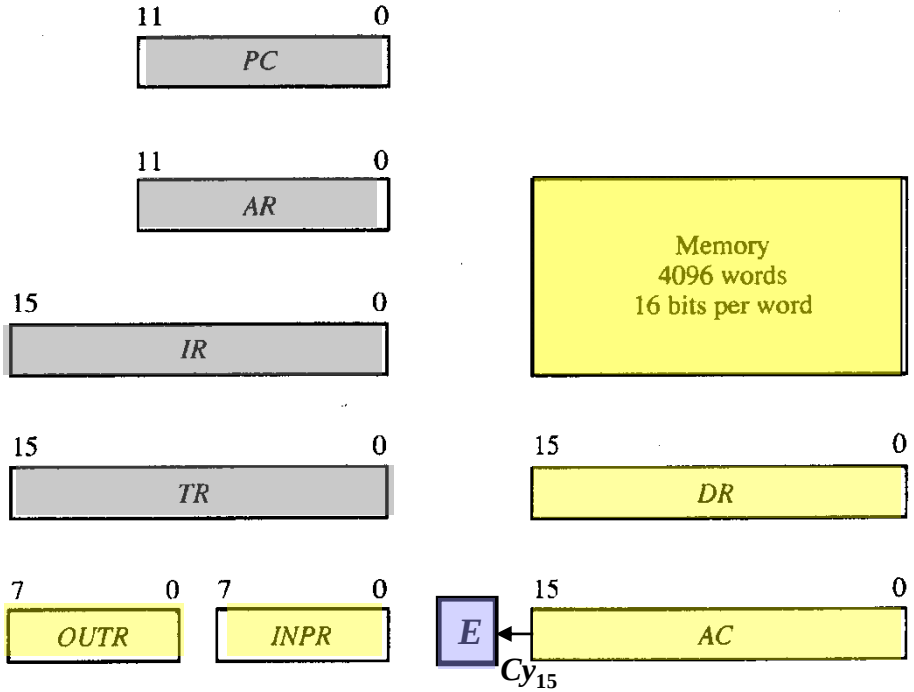


- Chaque mot de la partie données de la mémoire contient une opérande de 16 bits.
- L'accumulateur AC est un registre de processeur généralement utilisé pour stocker un opérande de l'opération à effectuer.

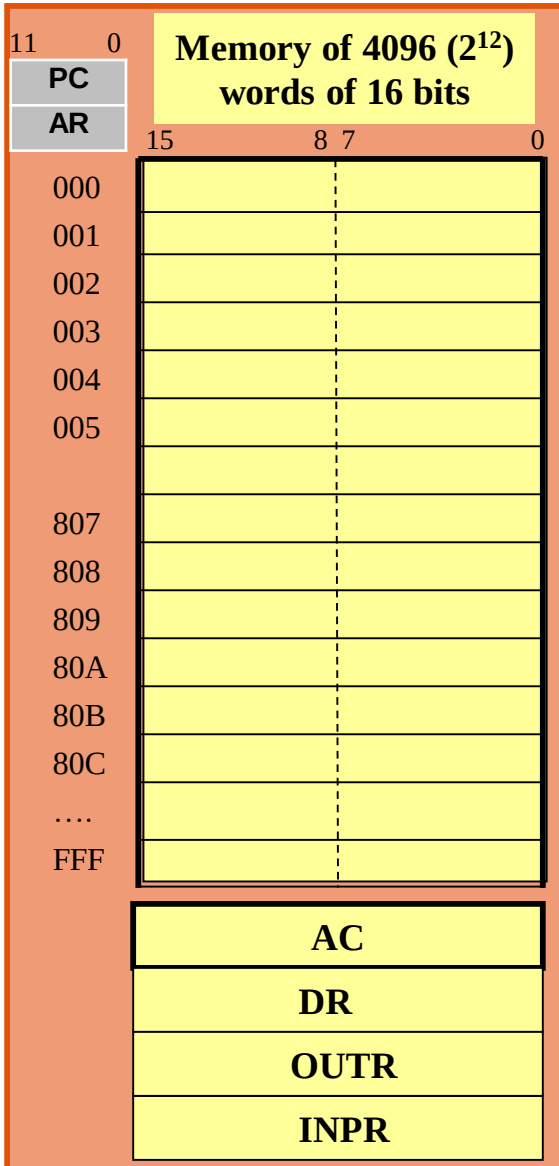


- Une instruction de programme est exécutée dans l'ordre séquentiel suivant:
  1. L'unité de commande lit le code d'instruction de 16 bits à partir de la partie du programme de la mémoire.
  2. La partie d'adresse du code d'instruction de 12 bits est ensuite utilisée pour lire l'opérande de 16 bits à partir de la partie de données de la mémoire.
  3. Le code d'opération à 4 bits est ensuite utilisé pour effectuer l'opération souhaitée sur l'opérande que vous venez de lire.

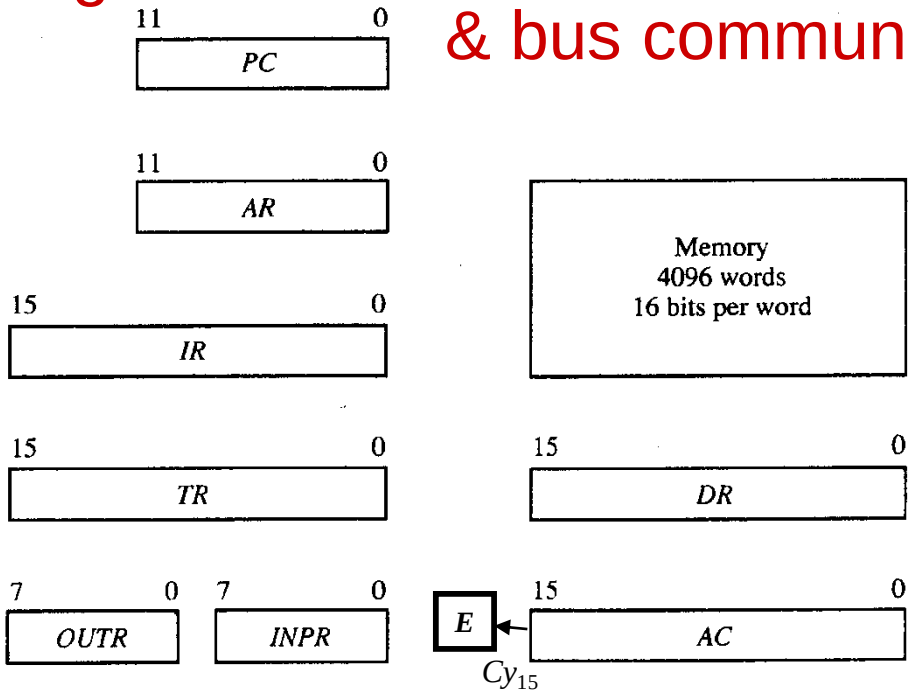
Registres de base de l'ordinateur et mémoire

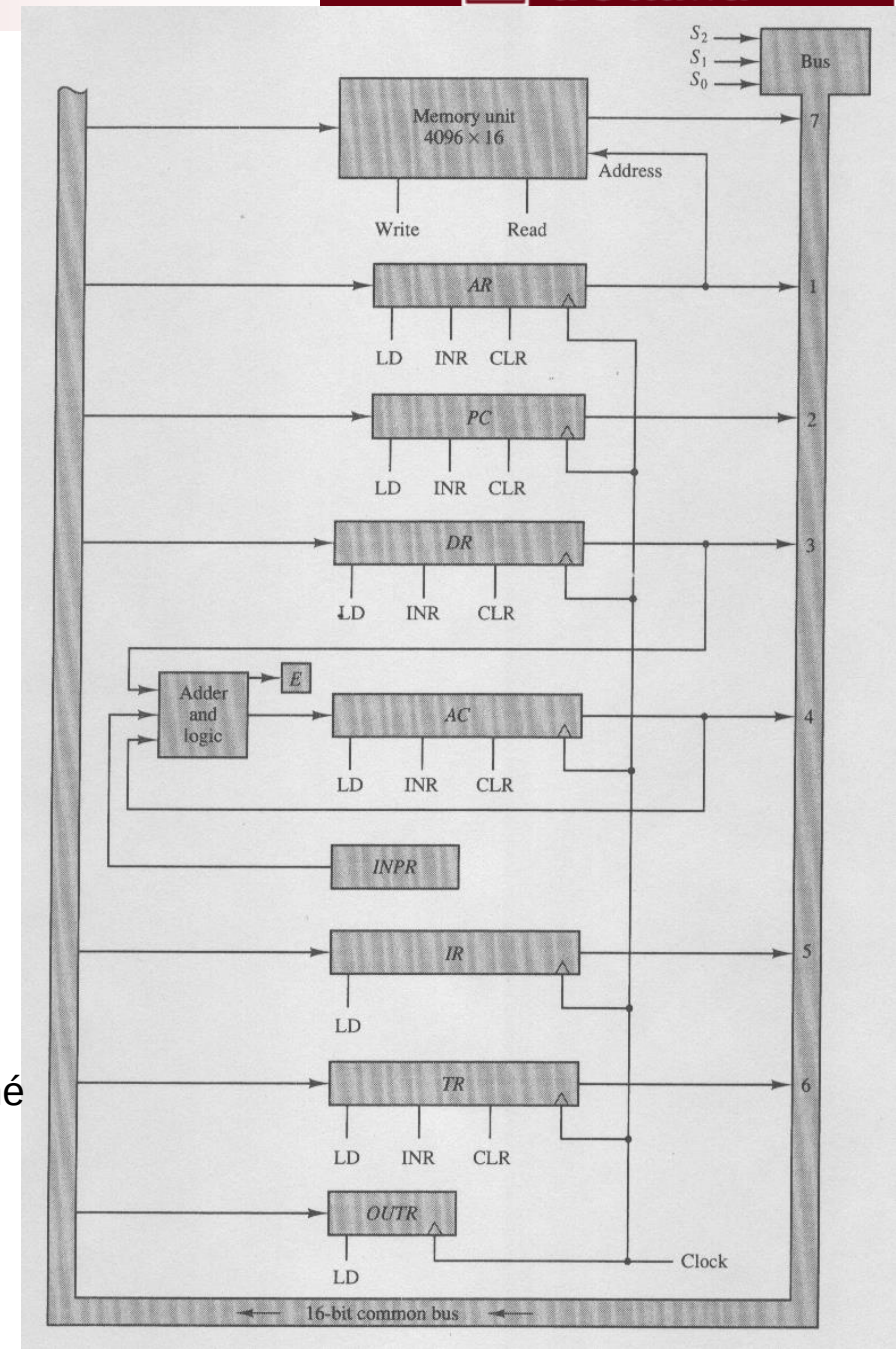


Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
TR	16	Temporary register	Holds temporary data
AR	12	Address register	Holds address for memory
PC	12	Program counter	Holds address of instruction
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character



Registres de base de l'ordinateur & bus commun





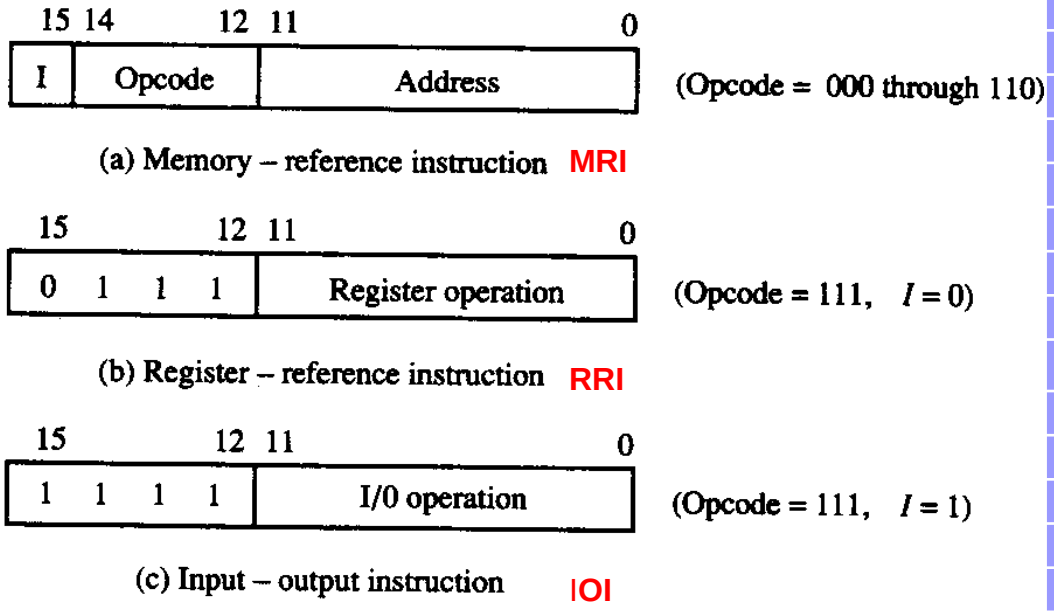
- Le contenu de tout registre peut être appliqué sur le bus et une opération peut être effectuée dans l'additionneur et le circuit logique au cours du même cycle d'horloge.
- La transition d'horloge à la fin du cycle transfère le contenu du bus dans le registre de destination désigné et la sortie du circuit additionneur et logique dans AC.
- Par exemple, les deux micro-opérations

  DR ← AC and AC ← DR

  peut être exécuté en même temps.

Liste d'instructions de base de l'ordinateur (Hex)

Il existe trois types d'instructions qui ont des formats différents:



Symbol	I=0	I= 1	Description
MRI			
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
RRI			
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
IOI			
INP		F800	Input character to AC
UT		F400	Output character from AC
SKI		F200	Skip on input flag
SKO		F100	Skip on output flag
ION		F080	Interrupt on
IOP		F040	Interrupt off

Modes d'adressage

- Adressage du registre (RRI / IOI)
  - Opérandes référencent les registres internes
  - Aussi appelé adressage inhérent (Motorola)
- Adressage immédiat (non implémenté ici)
  - L'opérande est constant et est contenu dans le mot d'instruction qui suit immédiatement le code opération
- Adressage direct (MRI)
  - L'opérande doit être extrait de l'emplacement de mémoire dont l'adresse est donnée par les 12 bits qui suivent le code opération dans le mot d'instruction
- Adressage indirect (MRI)
  - Instruction spécifie où se trouve l'adresse de l'opérande
- Adressage indexé (non implémenté ici)
  - Trouve l'adresse de données en utilisant un index (un décalage)
- Adressage relatif (non implémenté ici)
  - additionne un décalage à la valeur actuelle du PC

Notation utilisée dans les commentaires

- Nom du registre: indique un registre et son contenu
  - Exemple: AC fait référence au contenu de l'accumulateur AC
- → (->) La flèche droite indique une opération de transfert de données (<-> indique un échange de données)
  - Exemple: AC -> DR indique que le contenu de AC est transféré (copié) vers DR
- M[…] Contenu d'un emplacement de mémoire
  - Exemple: M[1234**H**] -> indique que le contenu de l'emplacement de mémoire **HEX**1234 est transféré vers AC (également: M[**$**1234] -> AC)
- M[M[…]] Adressage indirect – Les parenthèses internes spécifient une adresse mémoire contenant l'adresse de données
  - Exemple: AC -> M[M[1234H]] indique que le contenu de AC est transféré dans un emplacement de mémoire dont l'adresse est trouvée dans l'emplacement de mémoire d'adresse 1234H (syntaxe: STA @1234H)

Adressage inhérent (Adressage de registre)

RRI / IOI

- Toutes les données pour les instructions sont dans le CPU
- Les instructions avec mode d'adressage inhérent sont les suivantes:
  - parmi les plus rapides à exécuter
  - codé avec le moins de bits.
- Ce mode est également appelé Adressage de registre ou adressage implicite
- Tous les RRI et IOI de l'ordinateur de base utilisent le mode d'adressage inhérent

CLA ; 0 -> AC

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Opcode														
CLA =7800_H	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

INC ; AC <- AC+1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Opcode														
INC=7020_H	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0

Adressage immédiat

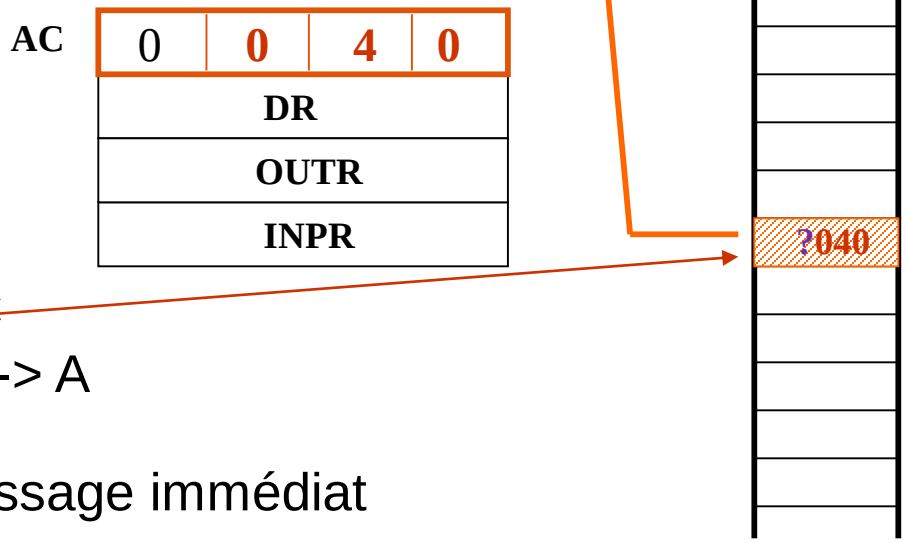
non mis en œuvre ici

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Opcode				Operand Value											
LDA #40H	?	?	?	?	0	0	0	0	0	1	0	0	0	0	0	0

Un argument est contenu dans les bits qui suivent immédiatement le code opération.

Il n'est pas implémenté dans cet ordinateur de base !!! Donc, l'opcode est marqué avec "?" dans l'exemple suivant..

Le contenu lsb de 12 bits du mot d'instruction (c'est-à-dire l'opérande) est transféré dans l'accumulateur AC



Exemples d'adressage immédiat

```
LDA   #64     ; Decimal 64 -> AC
LDA   #64H    ; Hexadecimal 64 -> A
```

- Le symbole # indique un adressage immédiat
- Il est facile d'oublier le #

Adressage direct

MRI

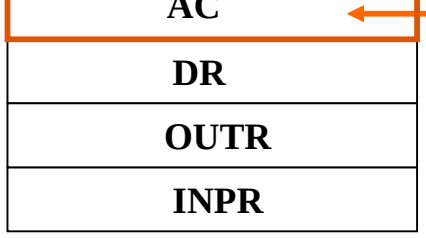
- L'instruction contient l'adresse de données
  - Adressage à un niveau
  - L'adresse de l'opérande est donnée par les 12 bits de poids faible suivant le code opération.
  - Instructions codées avec les opcodes 0 à 6 de la liste d'instructions du tableau de la diapositive 10 (avec **msb IR_{15}=0**) adresses d'accès $000–$FFF directement

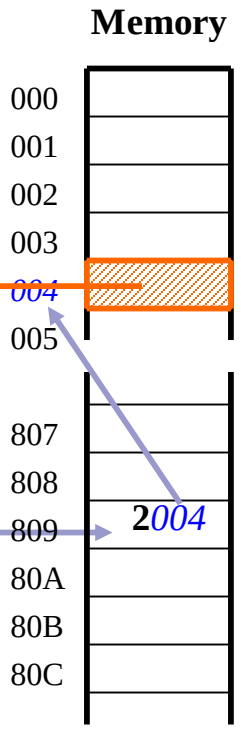
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Opcode				Operand Address											
LDA 004	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0

Exemple: **LDA 4 ;A <- M[4]**

2. Le contenu de 16 bits de l'emplacement de mémoire sélectionné est copié, comme spécifié par l'instruction, dans l'accumulateur AC.

1. Le 12 bits lsb du mot d'instruction (004) représente l'adresse de l'emplacement de mémoire où l'opérande est stocké





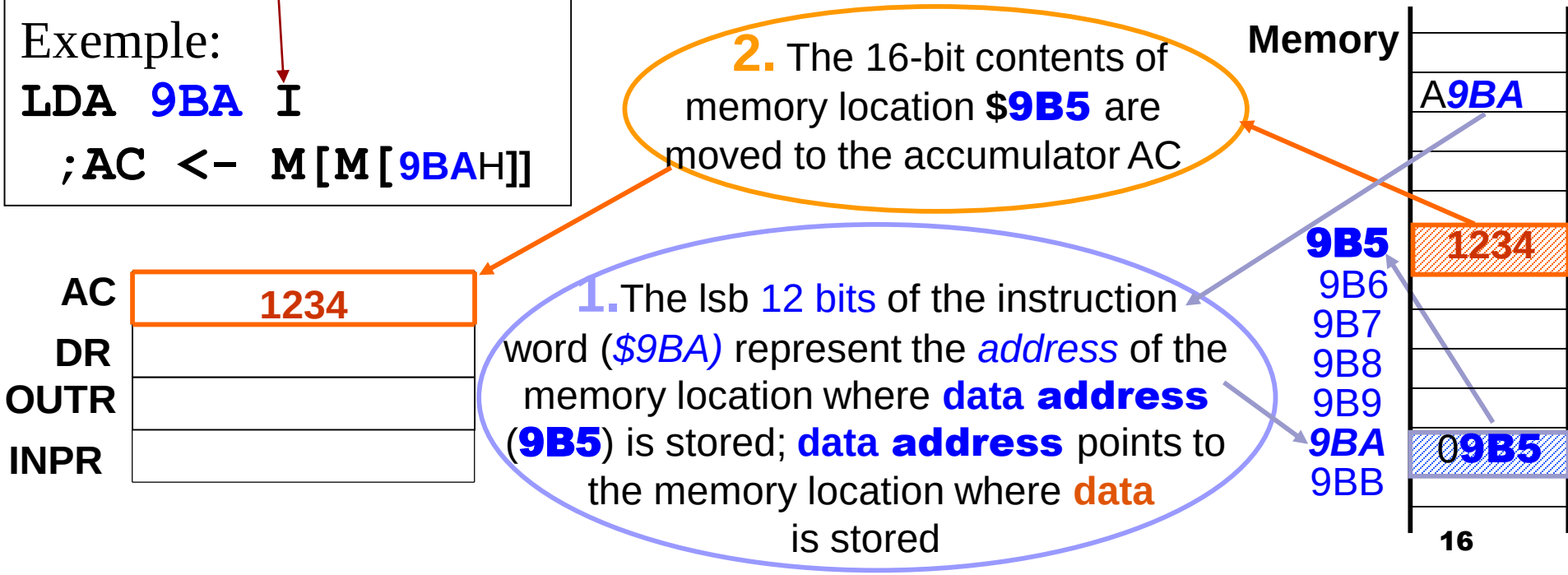
Adressage indirect

MRI

- Aussi appelé adressage par pointeur
- L'instruction contient l'adresse de l'emplacement mémoire (marqué d'un "**I**" ou "**@**"ou …) contenant l'adresse de données (pointeur)
- Mécanisme d'adressage à deux niveaux
  - Le premier niveau fourni par instruction donne l'adresse de la mémoire contenant une **adresse**
  - Le deuxième niveau est l'adresse qui spécifie où se trouvent les données
  - Instructions codées avec les codes d'opération 8 - E (msb IR_{15}=1) adresses d'accès $000–$FFF indirectement. En réalité, l'espace mémoire auquel ils peuvent accéder est de 64 kW (mots kilo)

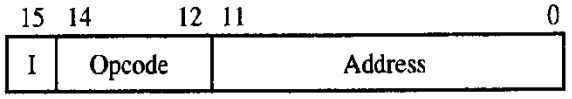
Exemple:

```
LDA  9BA  I
  ;AC <- M[M[9BAH]]
```

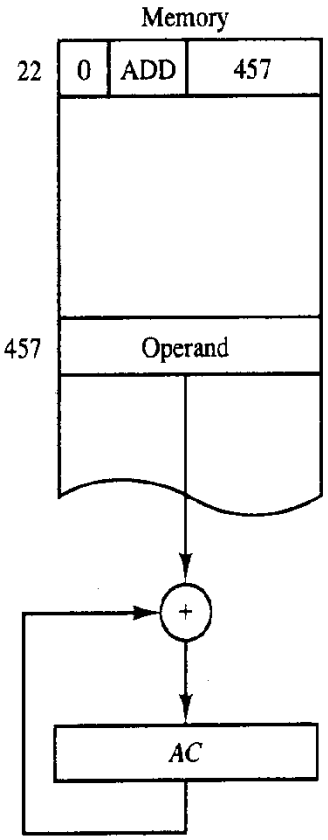


Adresse directe VS indirecte

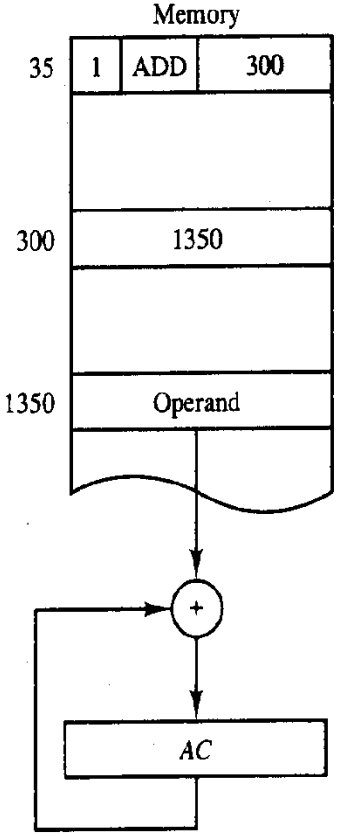
Le code d'instruction est composé d'un code d'opération à 3 bits, d'une adresse à 12 bits et d'un bit de mode d'adresse indirecte I (I = 0 pour une adresse directe et I = 1 pour une adresse indirecte).
La figure (b) montre un mode d'adresse directe (I = 0) avec un code d'opération d'une opération ADD.
Le code d'adresse 12 bits contient l'équivalent binaire de 457. ⇒ L'opérande se trouve à l'adresse 457 de la mémoire.
L'opérande est récupéré et ajouté au nombre stocké dans l'accumulateur AC.
La figure (c) montre un mode d'adresse indirecte (I = 1) avec un code opération d'une opération ADD.
Le code d'adresse de 12 bits contient l'équivalent binaire de 300. ⇒ L'adresse de l'opérande est extraite de l'adresse 300, qui contient l'équivalent binaire de 1350.
L'opérande est ensuite extrait de l'adresse 1350 et ajouté au nombre stocké dans l'accumulateur AC.



(a) Instruction format



(b) Direct address

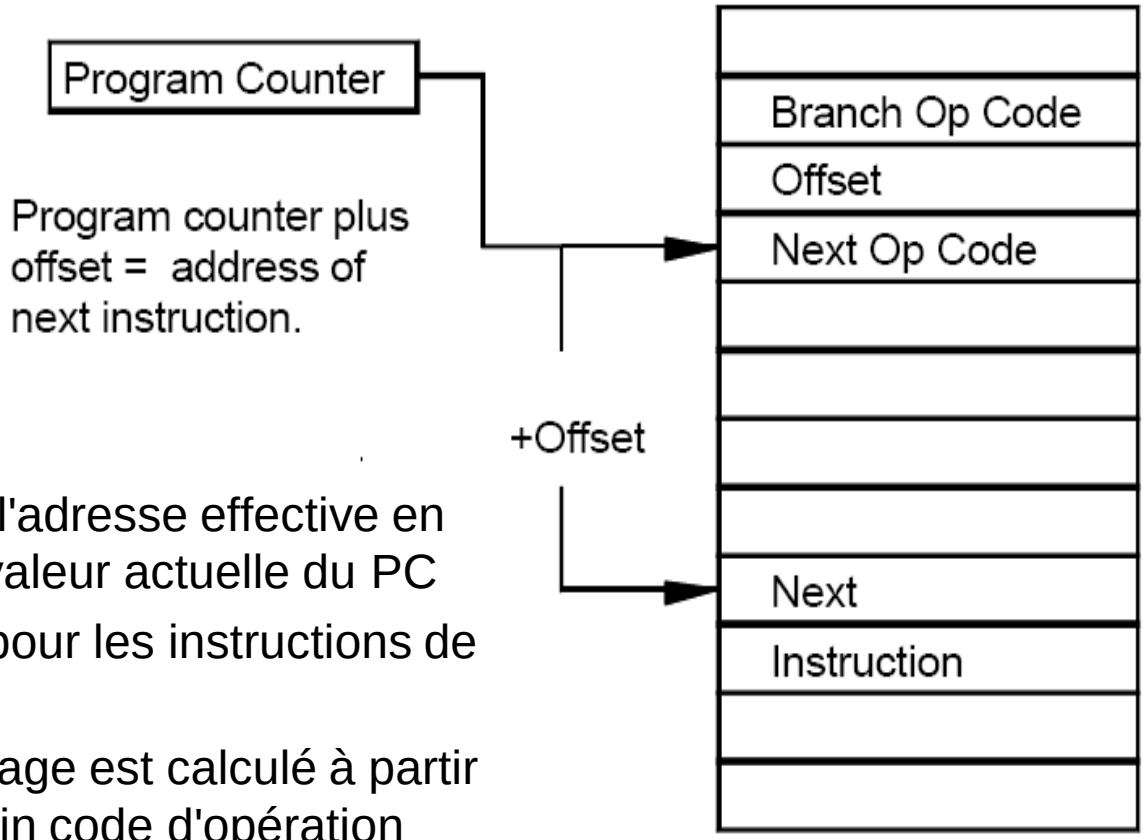


(c) Indirect address

Adressage relatif

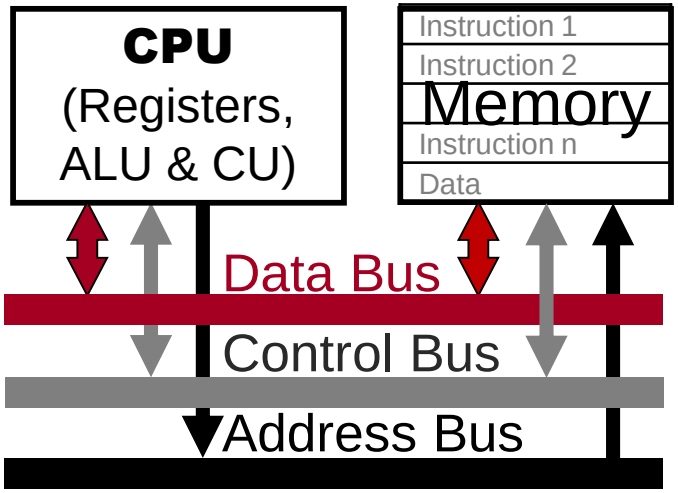
non implémenté ici

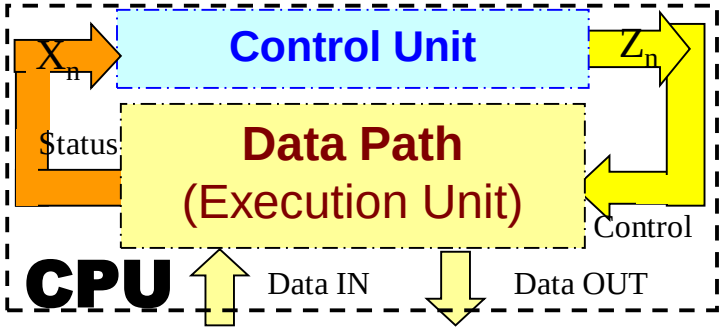
- L'adresse de l'opérande est appelée **l'adresse effective**.
- Par exemple, l'adresse effective dans la figure précédente (b) est 457, alors qu'elle est 1350 dans la figure (c).



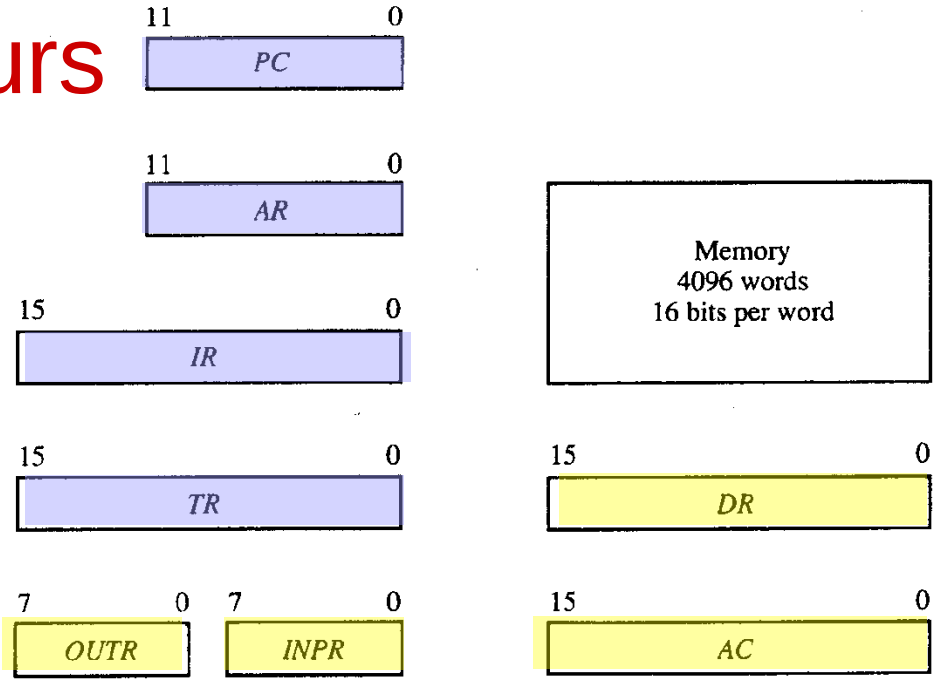
- L'adressage relatif calcule l'adresse effective en ajoutant un décalage à la valeur actuelle du PC
  - Utilisé principalement pour les instructions de branchement
  - Normalement, le décalage est calculé à partir de l'adresse du prochain code d'opération
- Non implémenté dans notre ordinateur de basic

Ordinateur de basic



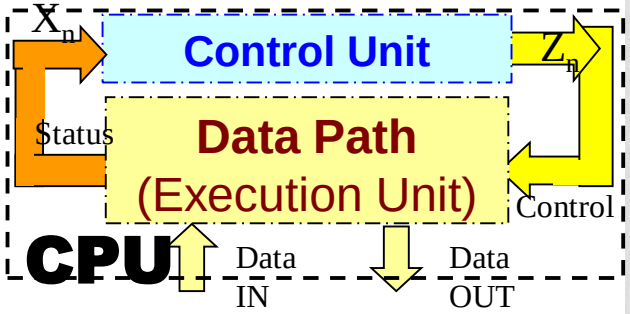


Registres d'ordinateurs

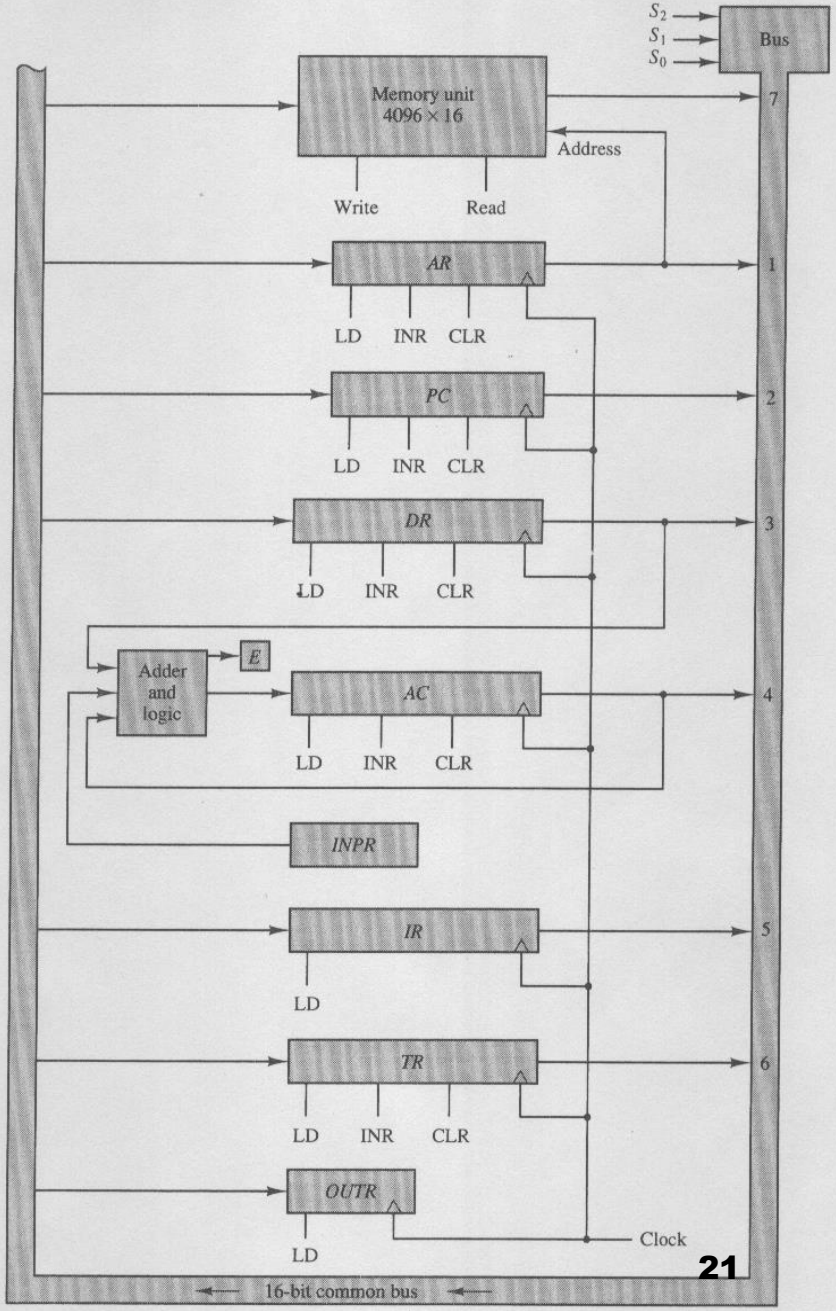


- La mémoire a une capacité de 4096 mots de 16 bits chacun.
- Comme le montre la figure 29 (a), chaque code d'instruction de 16 bits est composé de 12 bits d'adresse, d'un code d'opération de 3 bits et d'un bit pour spécifier une adresse directe ou indirecte.
- Le registre de données (DR) contient l'opérande lu dans la mémoire.
- L'accumulateur (AC) est un registre de traitement à usage général.
- Le registre d'instructions (IR) contient l'instruction lue dans la mémoire.
- Le registre temporaire (TR) conserve des données temporaires pendant le traitement
- Le registre d'adresse (AR) est un registre de 12 bits qui contient l'adresse du mot auquel on veut accéder dans la mémoire.
- Le compteur de programme (PC) est un registre de 12 bits qui contient l'adresse de l'instruction suivante à extraire de la mémoire après l'exécution de l'instruction en cours.
- INPR est un registre d'entrée contenant un code à 8 bits d'un caractère lu à partir d'un périphérique d'entrée. OUTR est un registre de sortie contenant un code à 8 bits d'un caractère à transférer à un périphérique de sortie.

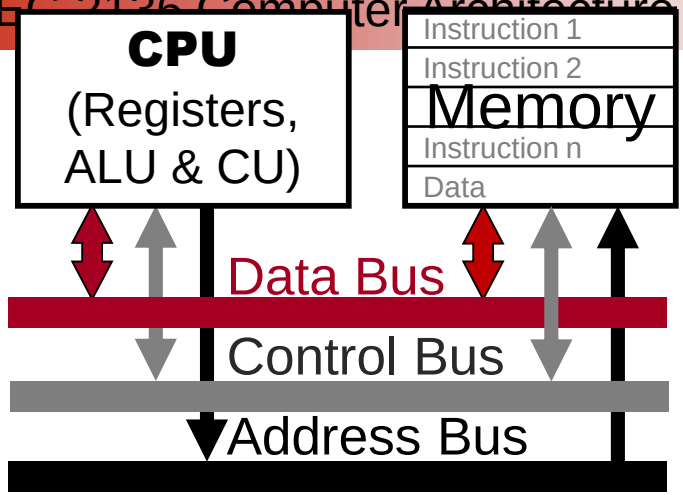
Chemin de données

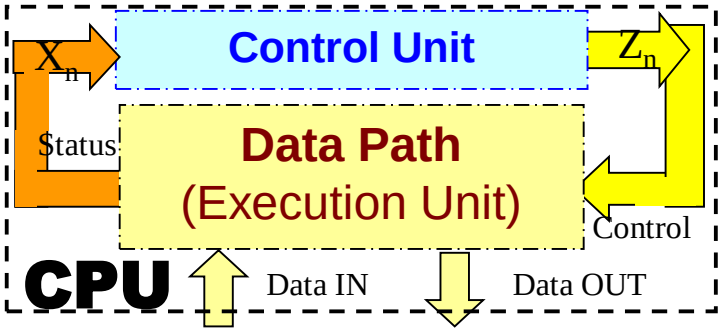


- Les lignes de bus communes sont connectées aux lignes d'entrée de données de chaque registre ainsi qu'aux entrées de données de la mémoire.
- Le registre particulier dont le bit d'entrée LD (charge) est activé reçoit les données du bus pendant l'impulsion d'horloge suivante.
- Le contenu du bus est chargé dans la mémoire lorsque son bit de contrôle d'écriture est activé.
- La mémoire place l'un de ses mots de 16 bits sur le bus lorsque son bit de contrôle de lecture est activé et que $S_2S_1S_0 = 111$.
- Deux des registres connectés au bus n'ont que 12 bits (AR et PC).
- Lorsque le contenu de AR ou PC est appliqué au bus 16 bits, les bus de 4 msb sont remis à "0".
- Lorsque AR ou PC charge le contenu du bus, seuls les 12 lbs sont transférés au registre.

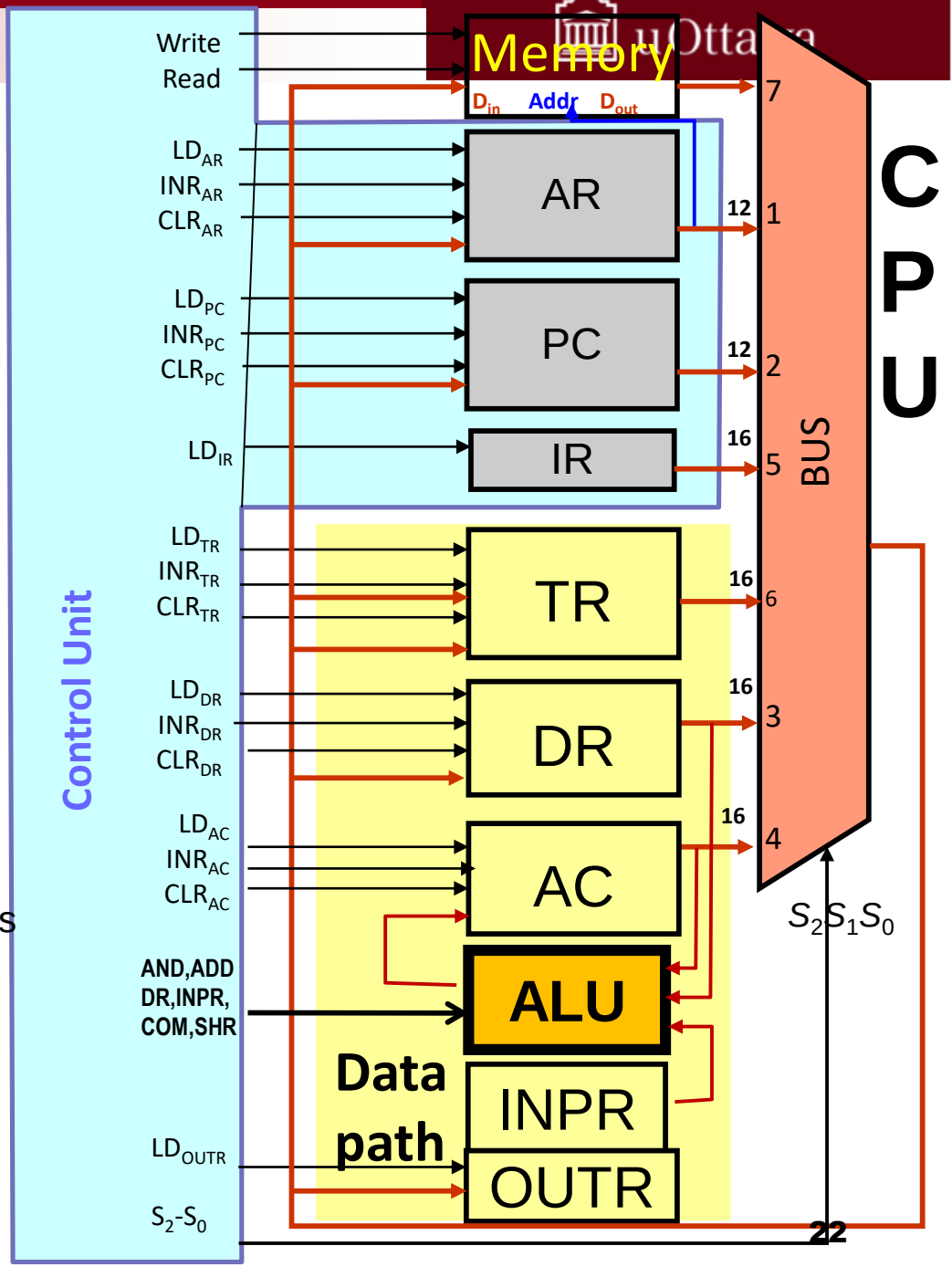


ORDINATEUR DE BASE



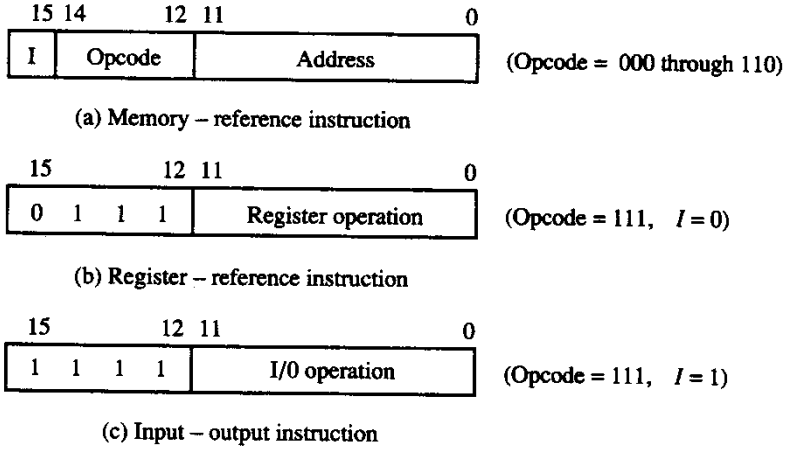


- L'ordinateur de base dispose de 8 registres, d'une unité de mémoire et d'une unité de contrôle, interconnectés via un bus commun.
- Les bits de sélection de bus S2, S1 et S0 déterminent la sortie du registre ou de la mémoire spécifique à placer sur les lignes de bus à tout moment.
- Le nombre le long de chaque entrée BUS de la figure suivante montre l'équivalent décimal de la sélection binaire requise.



uOttawa

Instructions d'ordinateur de base de base



Il existe 3 formats d'instruction.
L'opcode est composé de 3 bits ($I_{14}I_{13}I_{12}$)
000-110 = opérations à effectuer sur *AC*

- le reste des 12 bits (I_{11} - I_0) spécifie une adresse en mémoire:
- I_{15} = 0 => instruction avec adresse directe
- I_{15} = 1 => instruction avec adresse indirecte

111 & I_{15} = 0 => une référence de registre instr.
111 & I_{15} = 1 => une instruction entrée-sortie.
les 12 autres bits (I_{11} - I_0) sont utilisés pour spécifier le type d'opération à effectuer

Symbol	I=0	I= 1	Description
MRI			
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
RRI			
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
IOI			
INP		F800	Input character to AC
UT		F400	Output character from AC
SKI		F200	Skip on input flag
SKO		F100	Skip on output flag
ION		F080	Interrupt on
IOP		F040	Interrupt off

Liste d'instructions d'ordinateur de base (binaire)

D_7	Instr. Dec	Addressing Mode I_{15}=0	Addressing Mode I_{15}=1	op code I_{14}	I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
		MRI Direct Addr.	**MRI** Indirect Addr.				Memory				A	D	D	R	E	S	S	
D_7=0 **mem**	D_0	**AND**=$0addr	AND_i =$8(addr)	0	0	0	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_1	**ADD**=$1addr	ADD_i =$9(addr)	0	0	1	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_2	**LDA** =$2addr	LDA_i =$A(addr)	0	1	0	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_3	**STA** =$3addr	STA_i =$B(addr)	0	1	1	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_4	**BUN** =$4addr	BUN_i =$C(addr)	1	0	0	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_5	**BSA** =$5addr	BSA_i =$D(addr)	1	0	1	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_6	**ISZ** =$6addr	ISZ_i =$E(addr)	1	1	0	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
D_7=1 **reg**		**RRI**	**IOI**	**1**	**1**	**1**	C o d e				Extension							
	D_7	**CLA** =$7800	INP = $F800	1	1	1	**1**	0	0	0	0	0	0	0	0	0	0	0
	D_7	**CLE** =$7400	OUT = $F400	1	1	1	0	**1**	0	0	0	0	0	0	0	0	0	0
	D_7	**CMA** =$7200	SKI = $F200	1	1	1	0	0	**1**	0	0	0	0	0	0	0	0	0
	D_7	**CME** =$7100	SKO = $F100	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0
	D_7	**CIR** =$7080	ION = $F080	1	1	1	0	0	0	0	**1**	0	0	0	0	0	0	0
	D_7	**CIL** =$7040	IOF = $F040	1	1	1	0	0	0	0	0	**1**	0	0	0	0	0	0
	D_7	**INC** =$7020	n/a	1	1	1	0	0	0	0	0	0	**1**	0	0	0	0	0
	D_7	**SPA** =$7010	n/a	1	1	1	0	0	0	0	0	0	0	**1**	0	0	0	0
	D_7	**SNA** =$7008	n/a	1	1	1	0	0	0	0	0	0	0	0	**1**	0	0	0
	D_7	**SZA** =$7004	n/a	1	1	1	0	0	0	0	0	0	0	0	0	**1**	0	0
	D_7	**SZE** =$7002	n/a	1	1	1	0	0	0	0	0	0	0	0	0	0	**1**	0
	D_7	**HLT** =$7001	n/a	1	1	1	0	0	0	0	0	0	0	0	0	0	0	**1**

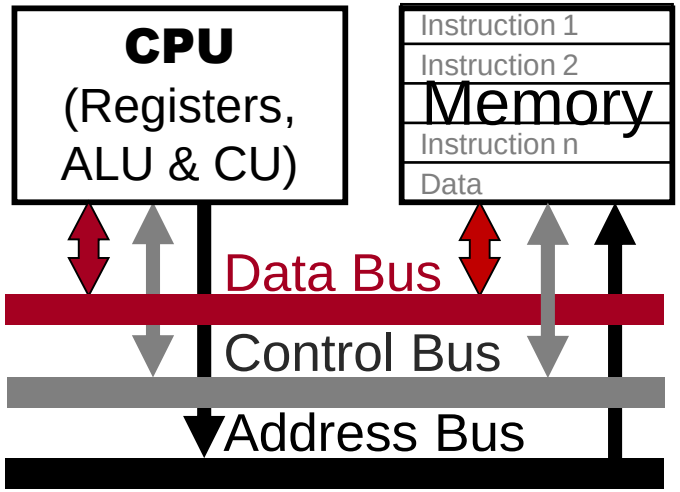
Binary encoding **One-hot** (*bit-per-state*) encoding

"addr" = 12 bit address of the operand
"(addr)" = address of the operand address

Complément d'instruction

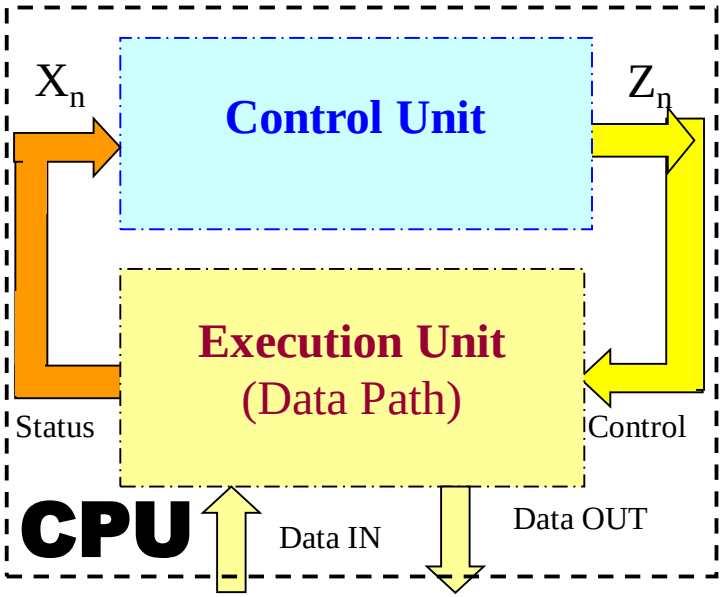
- Un jeu d'instructions informatiques est complet s'il peut être utilisé pour évaluer une fonction connue pour être calculable.
- Pour être complet, le jeu d'instructions doit contenir suffisamment d'instructions dans chacune des catégories suivantes:
  1. Arithmétique, logique et instructions de décalage.
  2. Instructions pour déplacer des informations vers et depuis les registres de mémoire et de processeur.
  3. Programmez les instructions de contrôle ainsi que les instructions qui vérifient les conditions d'état.
  4. Instructions d'entrée et de sortie
- Les instructions de contrôle du programme, telles que les instructions de branchement, permettent de modifier la séquence d'exécution du programme.
- Des instructions d'entrée et de sortie sont nécessaires pour la communication entre l'ordinateur et l'utilisateur (monde extérieur).
- Dans ce contexte, la liste d'instructions de l'ordinateur de base est complète car elle fournit suffisamment d'instructions dans chaque catégorie.

Conception descendante

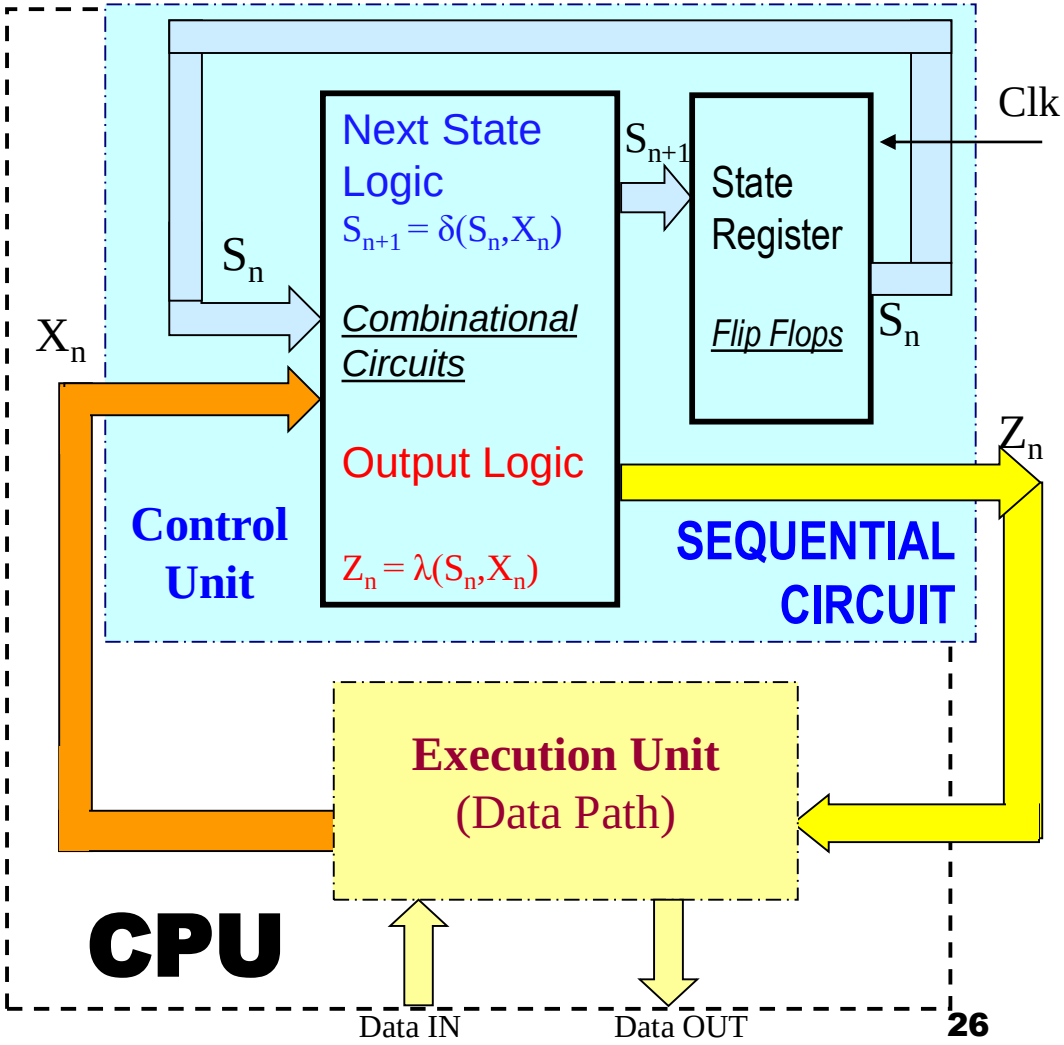


Central Processing Unit

CPU Block Diagram

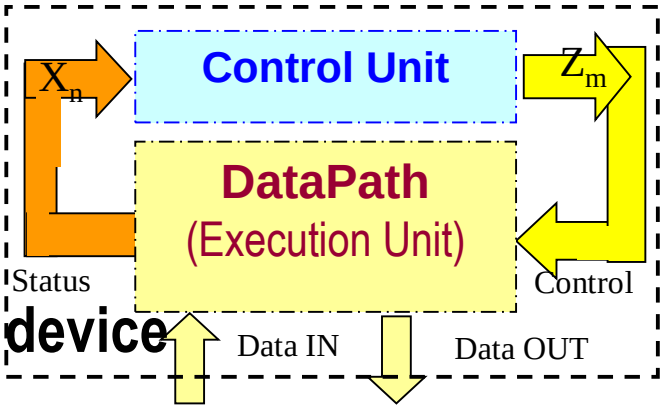


Unité de contrôle du CPU en tant que circuit séquentiel



Machines à états algorithmiques (Algorithmic State Machine, ASM)

- Le procédé ASM est utilisé pour résoudre un problème donné, en supposant que la cible est un dispositif électronique numérique, constitué d'un chemin de données et d'une unité de contrôle.



- Les cinq étapes principales de la méthodologie ASM sont les suivantes:

ASM # 1. À l'aide d'un pseudocode, décrivez l'algorithme à exécuter par l'appareil

ASM # 2. Convertir le pseudocode en diagramme ASM (avec RTL)

ASM # 3. Concevoir le chemin de données en fonction du diagramme ASM

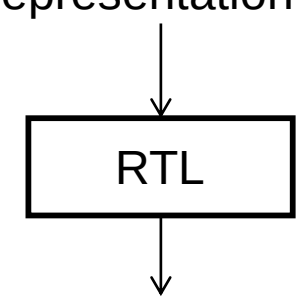
ASM # 4. Créer un diagramme ASM détaillé (équivalent à FSM) avec les signaux de contrôle qui doivent être générés par l'unité de contrôle pour diriger le chemin de données

ASM # 5 Concevoir la logique de contrôle sur la base du diagramme ASM détaillé

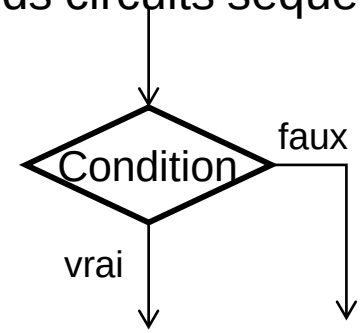
- Si elle est suivie correctement, la méthode ASM produit une conception matérielle de manière systématique et logique
  - Très robuste et facilement modifié
  - Affiné par itérations de pseudocodes

Constructions d'organigrammes ASM(ASM#2)

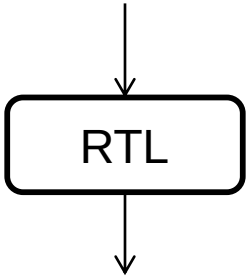
- À l'origine, les constructions ASM consistaient en des boîtes d'état, des boîtes de conditions et des boîtes de sortie conditionnelles
  - Les zones de sortie conditionnelles (Mealy FSM) sont difficiles à représenter sous forme de graphique ASM, sous forme de texte ou de tableau et permettent des transferts de données conditionnels (RTL)… peuvent être évités
- Les graphiques textuels ou tabulaires ASM ont remplacé les organigrammes en raison de leur facilité de modélisation et de la représentation de grands circuits séquentiels synchrones.



Boîte d'état



Boîte de condition



Boîte de sortie conditionnelle

Règles de l'organigramme ASM(ASM#2)

- Certaines règles doivent être respectées lors de la conception ASM:
  1. Les boîtes d'état ne doivent contenir que des instructions RTL, des signaux de contrôle et des instructions de transfert d'état entre parenthèses.
  2. Toutes les opérations dans une zone d'état doivent pouvoir être exécutées simultanément dans un cycle d'horloge.
  3. Si les opérations dans deux boîtes d'état consécutives peuvent être exécutées dans le même cycle d'horloge, les deux boîtes d'état peuvent être combinées en une seule boîte d'état;
  4. Les boîtes de conditions ne doivent contenir que des requêtes simples pouvant être évaluées à l'aide d'une logique purement combinatoire.
  5. Un nouveau registre doit être attribué à chaque nom unique dans l'ensemble d'instructions RTN;
  6. Pour chaque instruction registre de transfert, il doit exister un chemin entre les registres source et de destination (si une transformation a lieu pendant le transfert, un dispositif combinatoire, tel qu'un additionneur ou une unité ALU, doit être inséré dans le chemin entre la source et les registres de destination);
  7. S'il existe plusieurs chemins menant à un dispositif ou registre combinatoire, un multiplexeur ou une mémoire tampon à trois états doit être utilisé;
  8. Pour chaque requête binaire simple dans une zone de condition, un périphérique ou un signal d'état combinatoire doit être utilisé.
  9. Enfin, des entrées de signal de commande doivent être rattachées à chaque registre et multiplexeur afin que les transferts de registre puissent être contrôlés avec précision.

Exemple de conception ASM

- Le chemin de données d'un coprocesseur dédié possède trois registres AR, BR et CR de 16 bits..
- Lorsque le coprocesseur est déclenché par un début de signal, il effectue les opérations suivantes:
  - Transférez deux nombres signés de 16 bits (fournis par le microcontrôleur sur deux bus de données DB-A et DB-B en représentation complément à 2) vers AR et BR.
  - Si le nombre dans AR est négatif, divisez le nombre dans AR par 2 et transférez le résultat dans le registre CR.
  - Si le nombre dans AR est positif mais non nul, multipliez le nombre dans BR par 2 et transférez le résultat dans le registre CR.
  - Si le nombre dans AR est zéro, mettez le registre CR à 0.
  - Signalez la fin du traitement en définissant un drapeau terminé.

Diagramme bloc #1

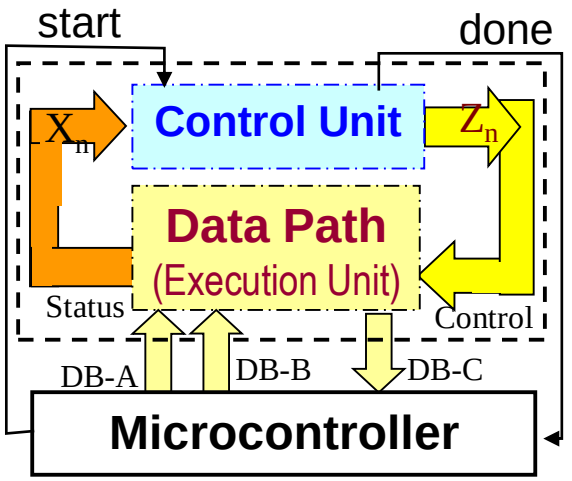
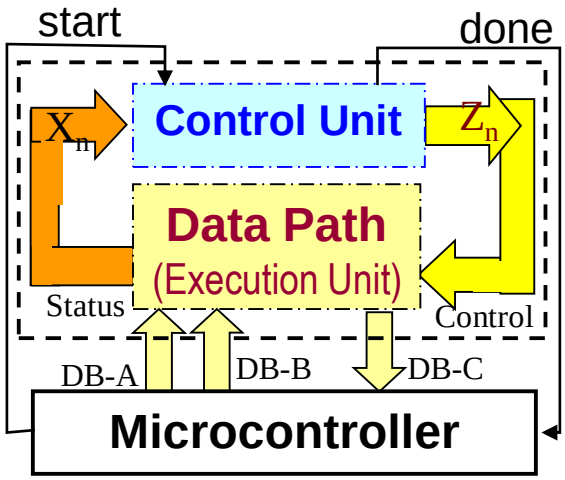
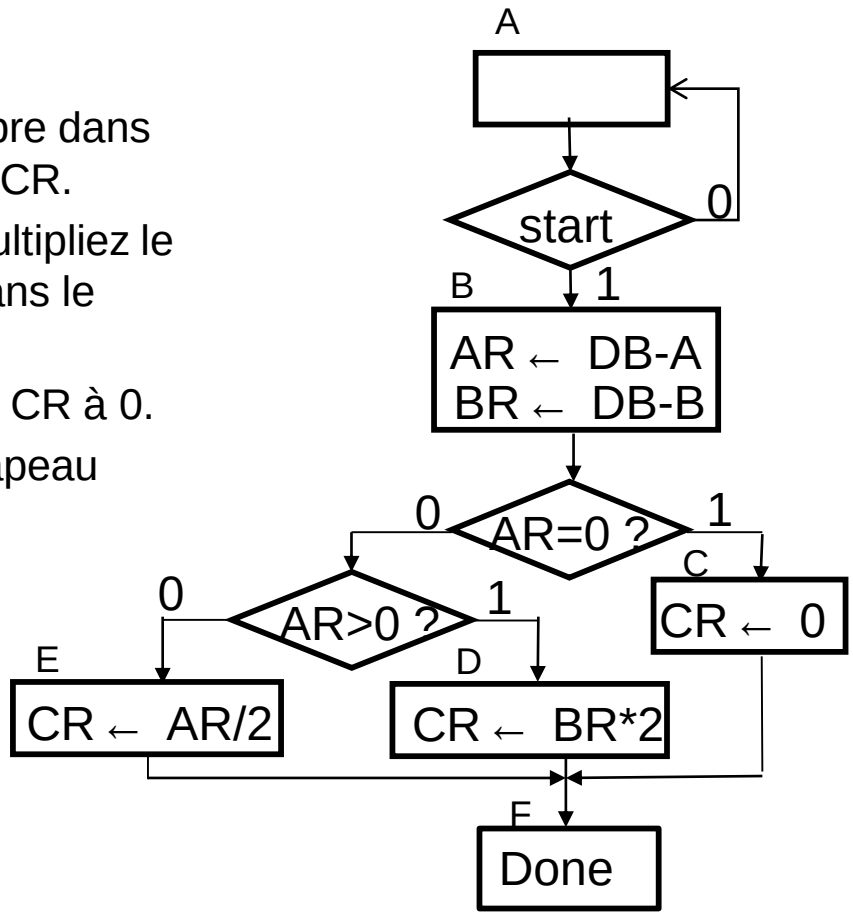


Diagramme bloc #1

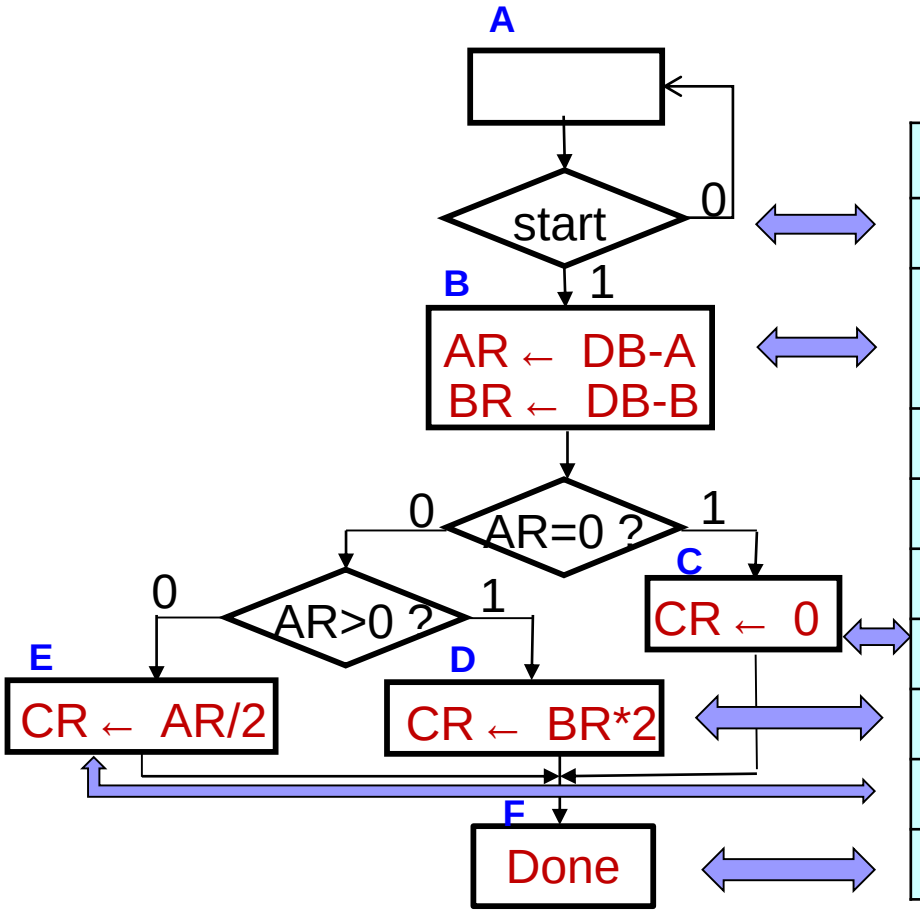
- Transférez deux nombres signés de 16 bits (fournis par le microcontrôleur sur deux bus de données DB-A et DB-B en représentation de complément à 2) vers AR & BR.
- Si le nombre dans AR est négatif, divisez le nombre dans AR par 2 et transférez le résultat dans le registre CR.
- Si le nombre dans AR est positif mais non nul, multipliez le nombre dans BR par 2 et transférez le résultat dans le registre CR.
- Si le nombre dans AR est zéro, mettez le registre CR à 0.
- Signalez la fin du traitement en définissant un drapeau terminé.



Organigramme ASM # 2 (RTL de haut niveau)



Organigramme ASM # 2 (RTL de haut niveau)

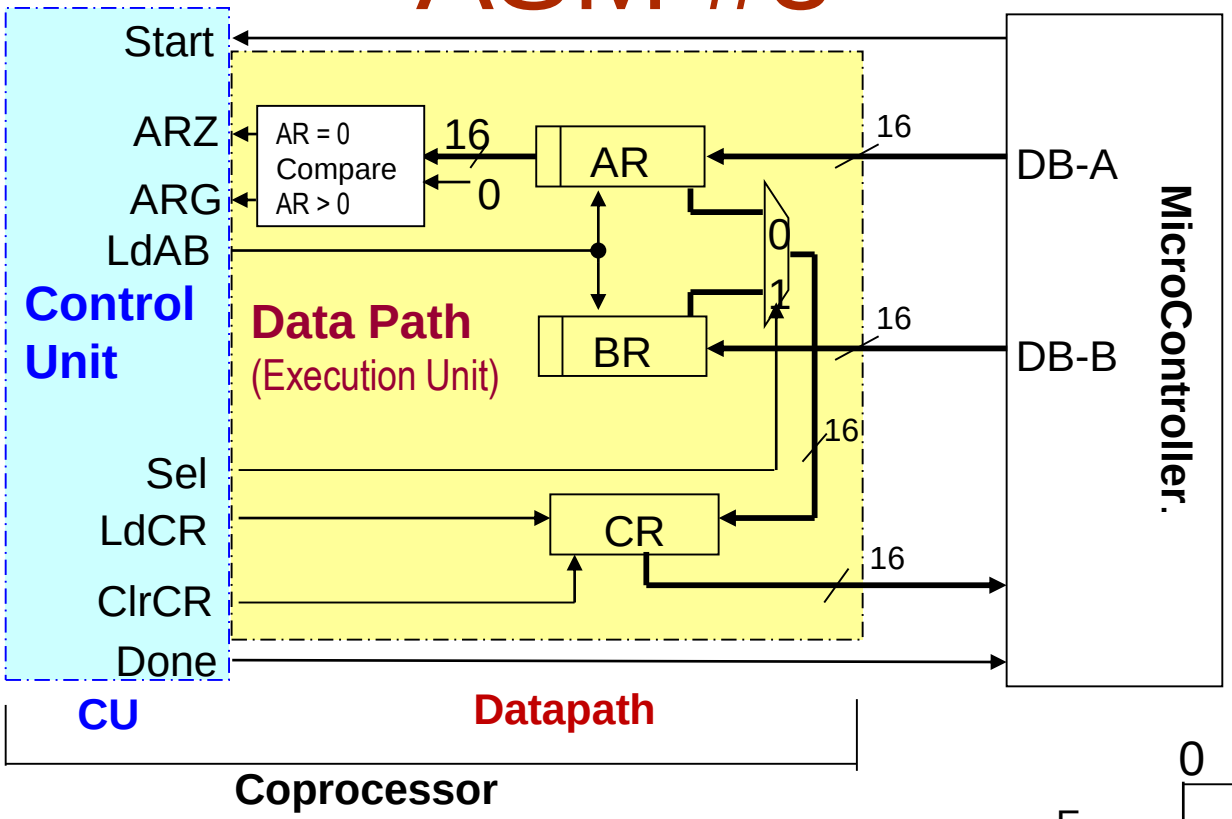


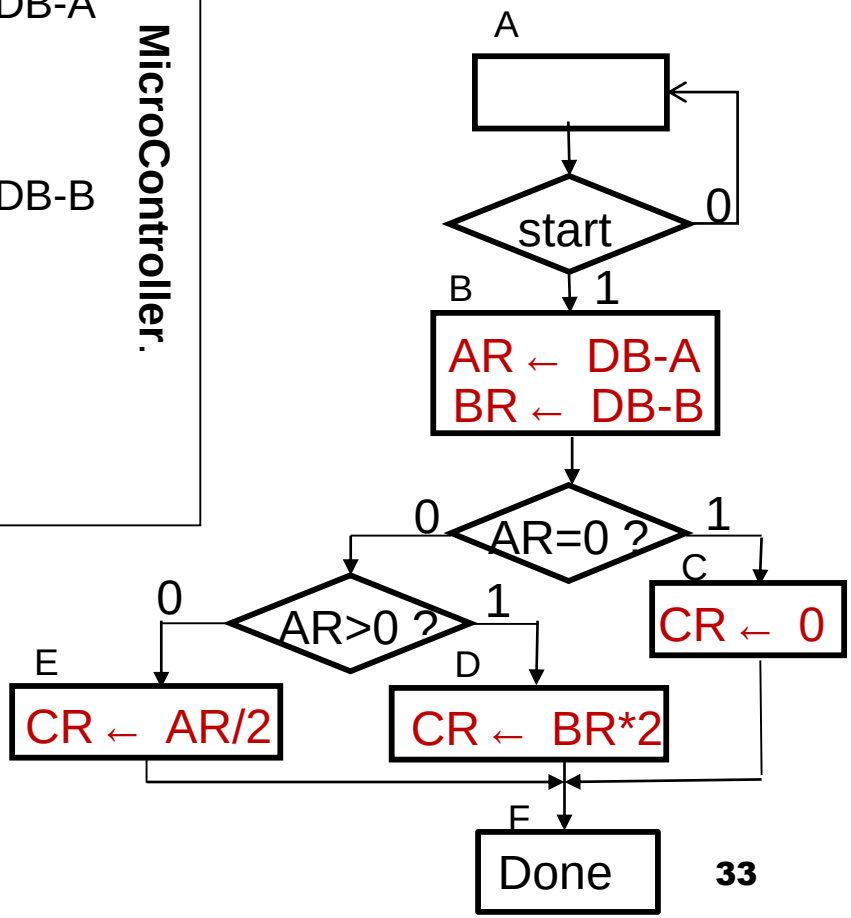
RTL

	$Z_n = \lambda(S_n)$	$S_{n+1} = \delta(S_n, X_n)$
A Start:		SC←B
B:	AR← DB-A BR←DB-B	
B (AR=0):		SC←C
B (AR≠0)(AR>0):		SC←D
B (AR≠0)(AR≤0):		SC←E
C:	CR←0	SC←F
D:	CR←BR*2	SC←F
E:	CR←AR/2	SC←F
F:		SC←A

Chemin de données ASM #3

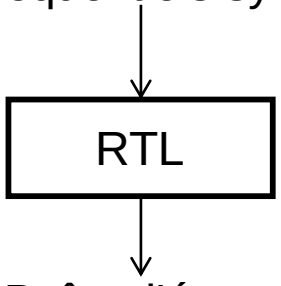
Déterminez les unités fonctionnelles (registres, bus, ALU) nécessaires pour exécuter les micro-opérations décrites dans l'organigramme ASM de haut niveau.



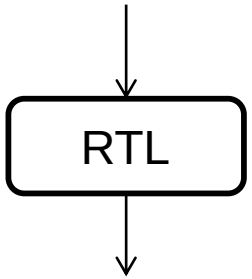


Chemin de données (ASM#2)

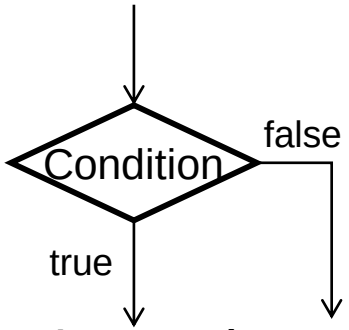
Les graphiques textuels ou tabulaires ASM remplacent les organigrammes en raison de leur facilité de modélisation et de la représentation de grands circuits séquentiels synchrones.



Boîte d'état

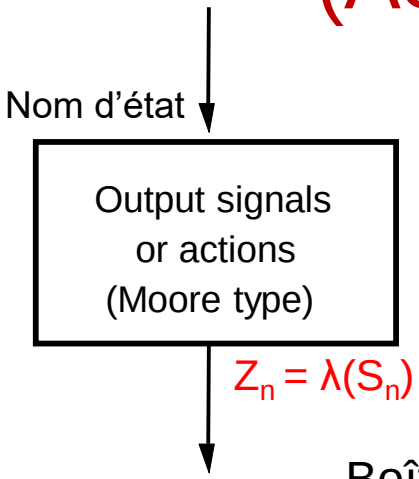


Boîte de sortie conditionnelle

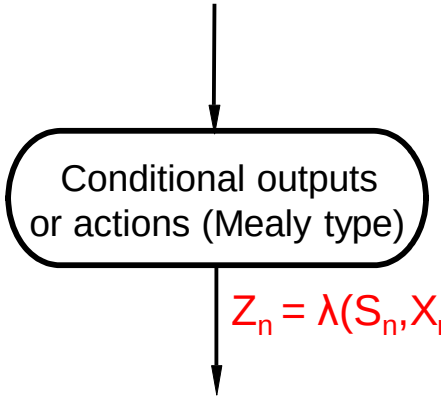


Boîte de décision

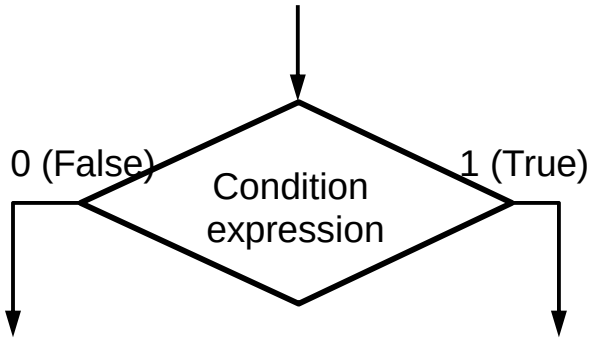
Diagramme détaillé (ASM#4)



Boîte d'état



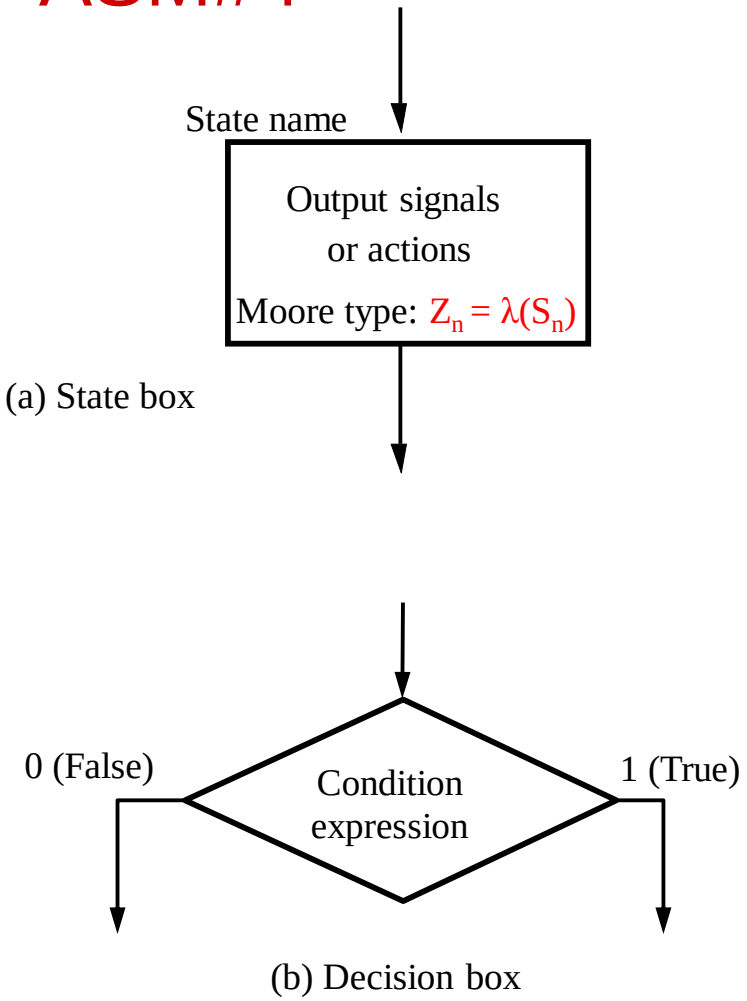
Boîte de sortie conditionnelle

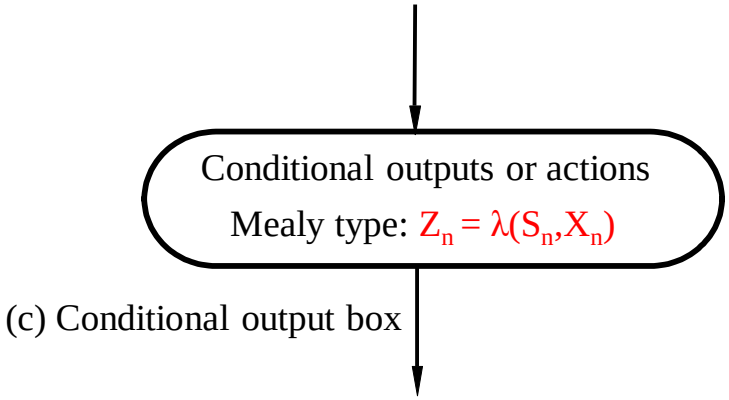


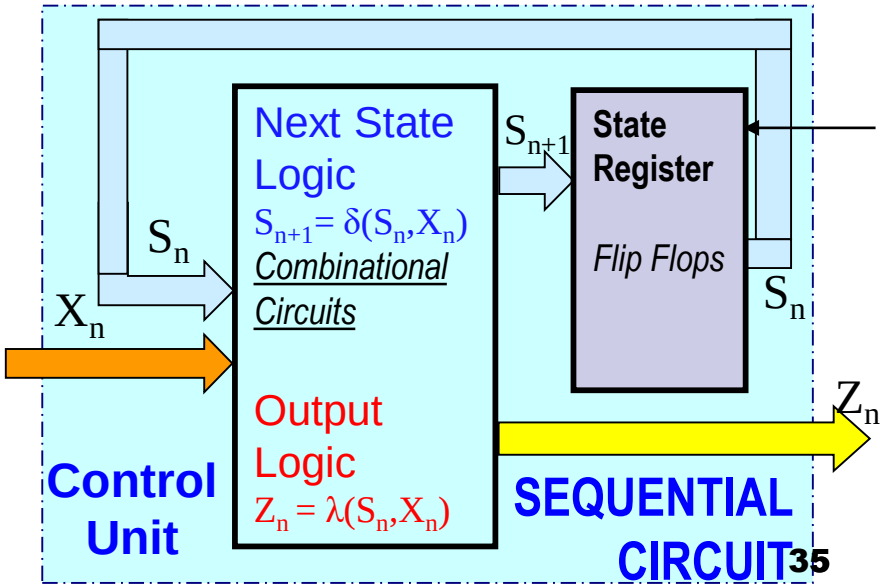
Boîte de décision

34

Symboles utilisés dans les graphiques détaillés ASM#4







Machines à états algorithmiques (ASM)

Les principales étapes de la méthodologie ASM sont les suivantes:

- Créer un algorithme, en utilisant un pseudocode, pour décrire le fonctionnement souhaité du périphérique
- Convertir le pseudocode en un graphique ASM (la brique de base = bloc ASM)
- Concevoir le chemin de données basé sur le graphique ASM
- Créer un graphique ASM détaillé basé sur le chemin de données
- Concevoir la logique de contrôle sur la base du diagramme ASM détaillé

La combinaison du chemin de données et de la logique de contrôle constitue le système logique réel qui résoudra le problème original.

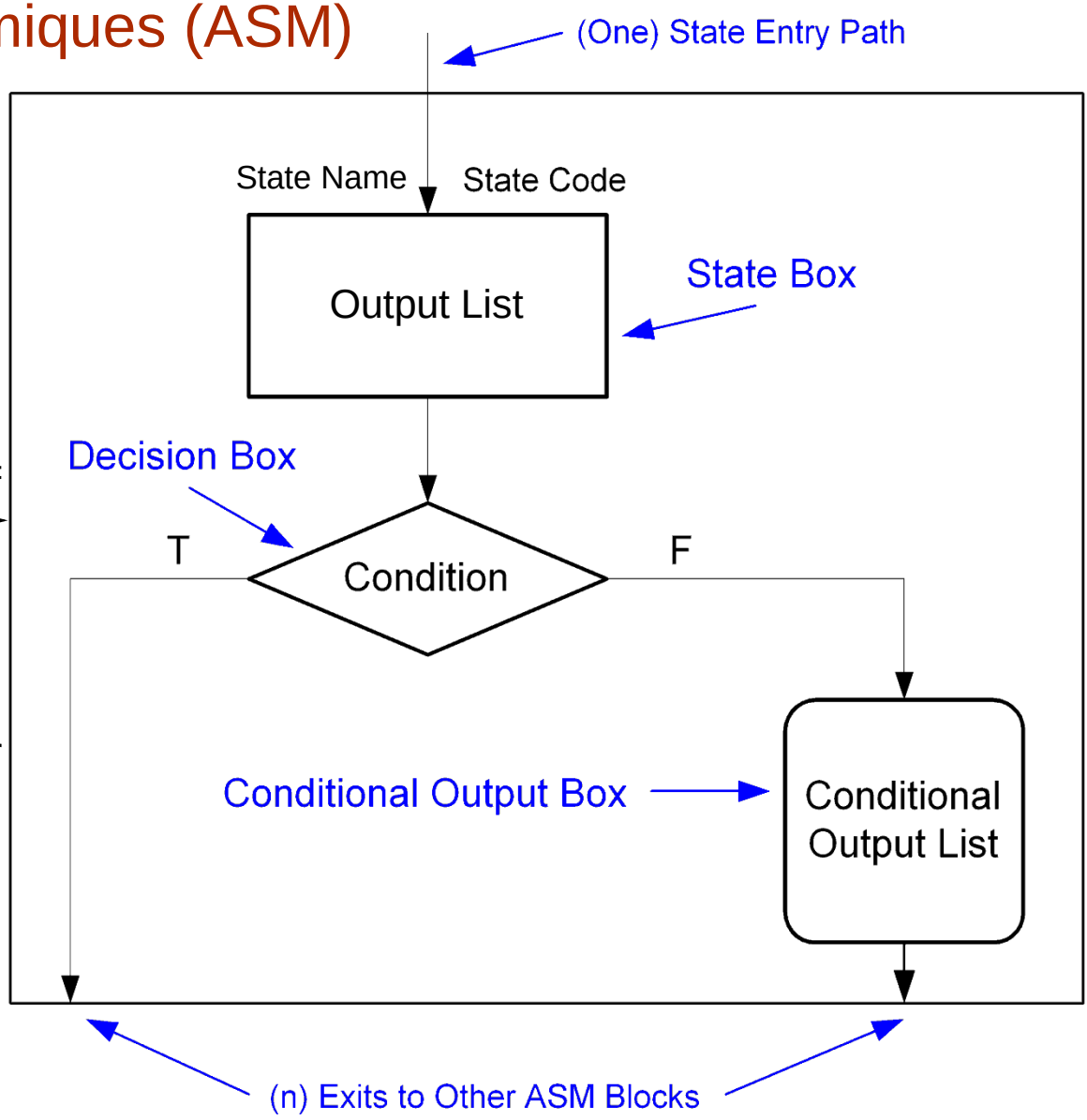
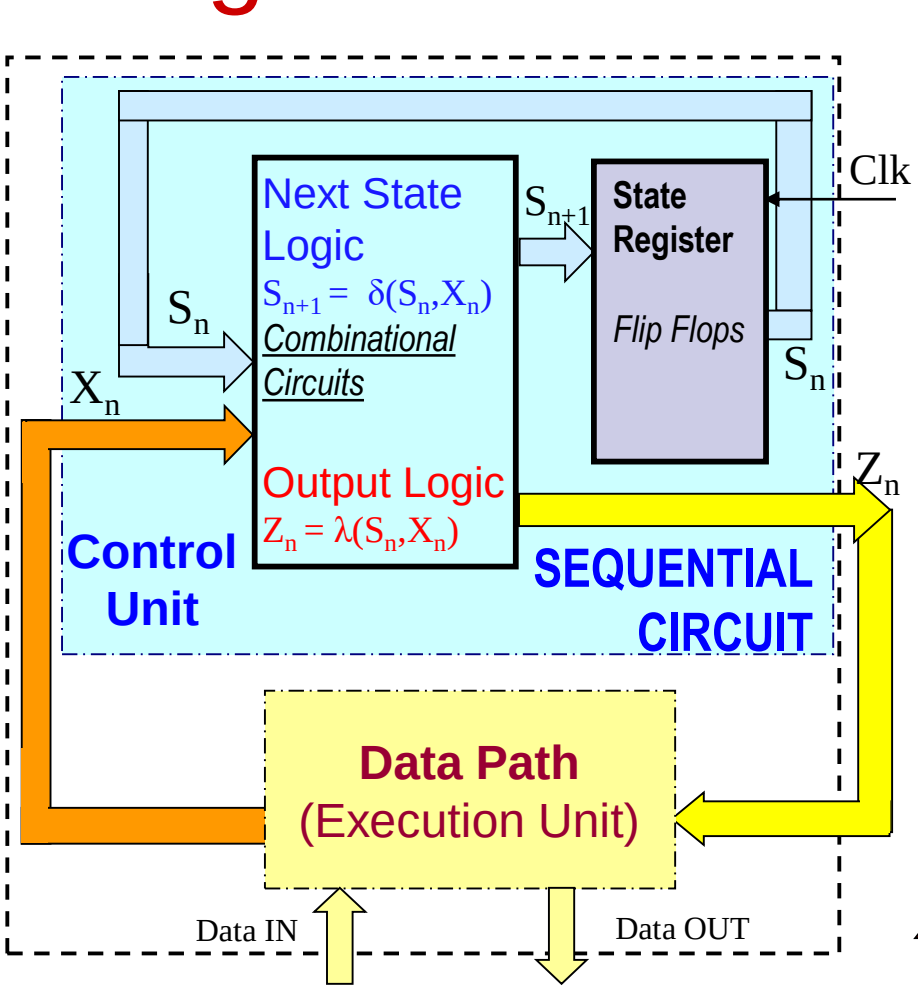
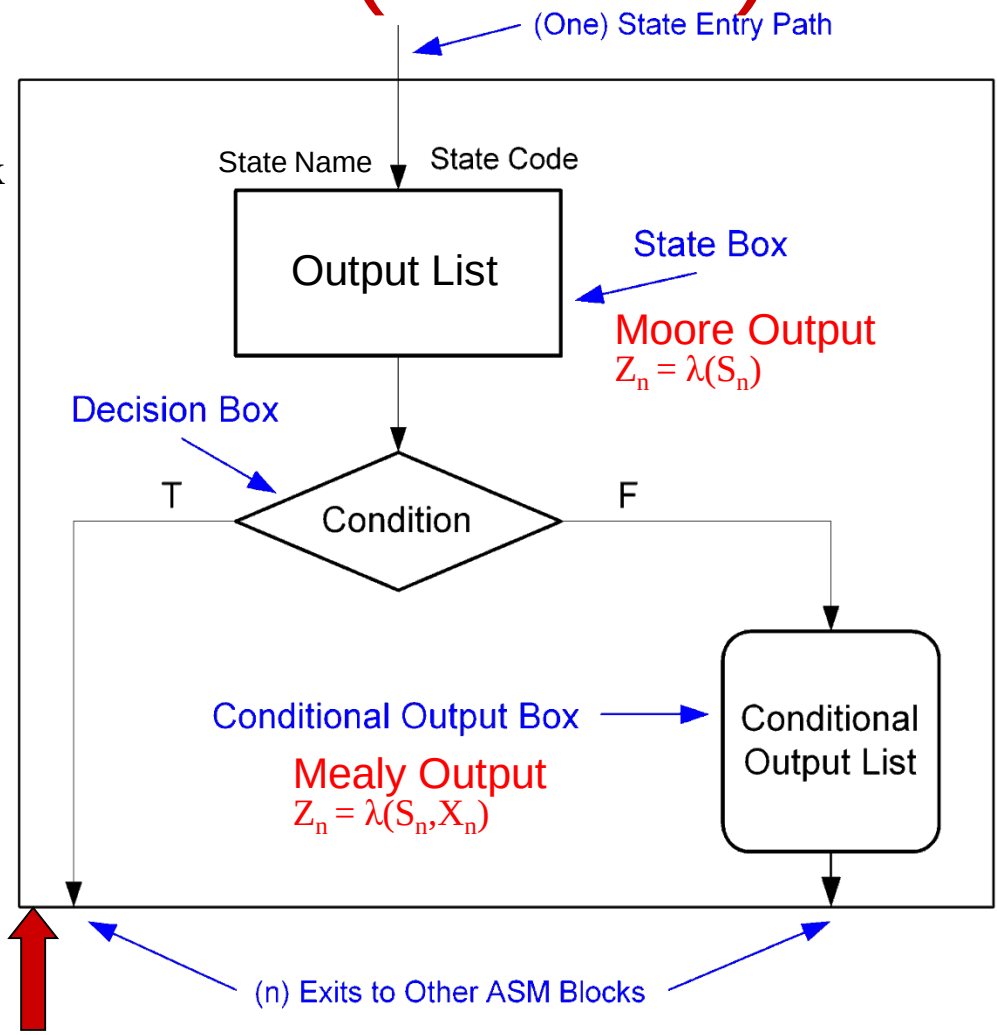


Diagramme détaillé ASM (ASM#4)





La brique de base = **bloc ASM**

La combinaison du **chemin de données** et de l'**unité de contrôle** constitue le système logique qui résoudra le problème initial.

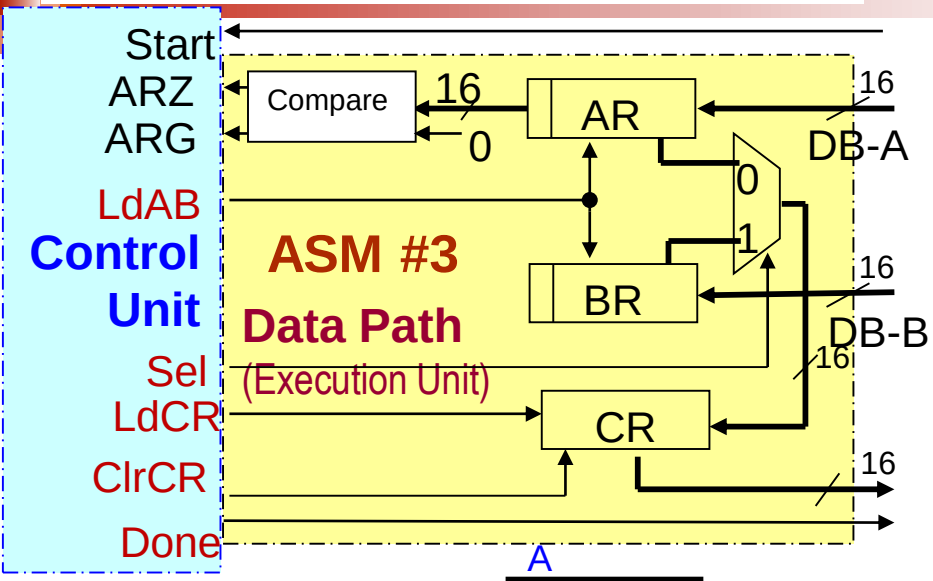
37

Règles du diagramme détaillé ASM (ASM # 4)

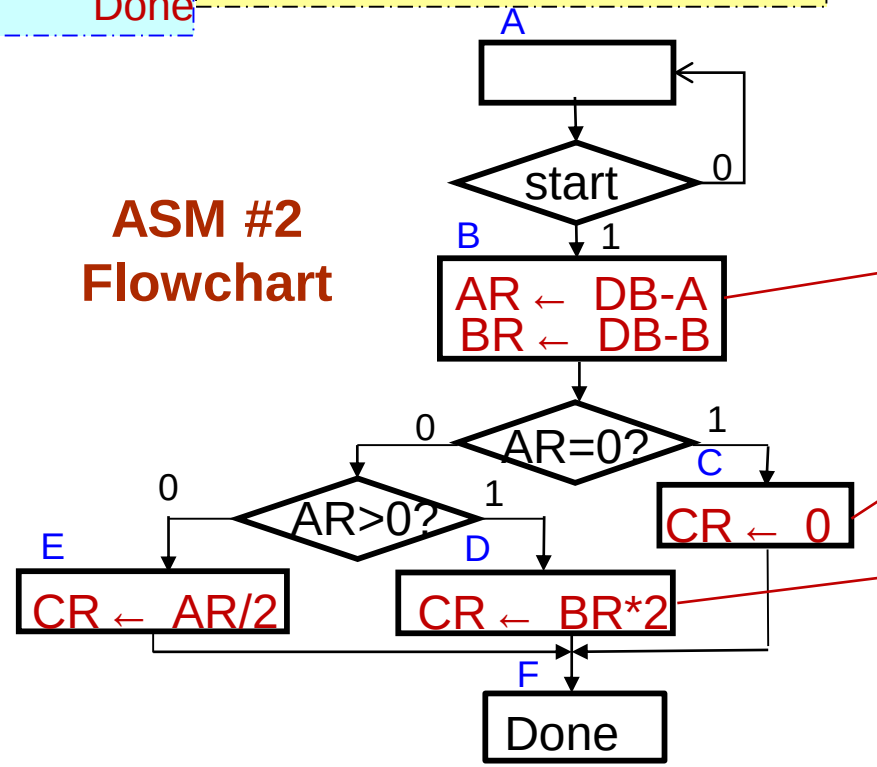
- Certaines règles doivent être respectées lors de la conception détaillée de cartes ASM
  1. Pour chaque case du diagramme ASM, il devrait y avoir une case d'état correspondante dans le diagramme ASM détaillé
  2. Pour chaque opération RTL dans une boîte d'état d'un diagramme ASM, la boîte d'état correspondante du diagramme ASM détaillé doit contenir une liste de sortie des signaux de contrôle appliqués au chemin Ces signaux de contrôle sont générés par l'unité de contrôle (CU) via sa fonction de sortie

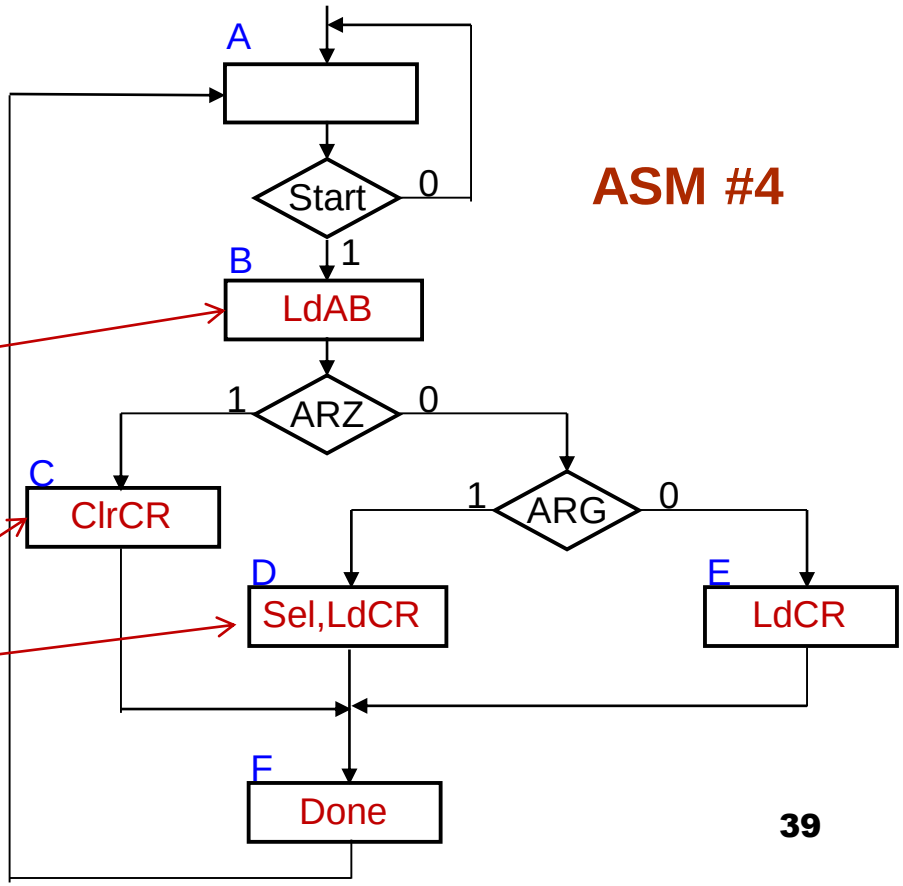
     Z = λ(S,X) - car CU est un circuit séquentiel.
     - Tous les signaux de commande sont supposés avoir la valeur "1" si spécifiée et la valeur "0" si non spécifiée.
  3. Pour chaque case de condition du diagramme ASM, il doit y avoir une case de condition correspondante dans le diagramme ASM détaillé
     - Chaque case de condition du diagramme ASM détaillé doit contenir une combinaison logique de signaux d'état (provenant du chemin de données) qui implémente la requête en logique combinatoire dans la case de condition de diagramme ASM correspondante.
- Portez une attention particulière à la boîte d'état et aux mappages de boîtes conditionnelles!
- La logique de contrôle est le cœur du circuit numérique et constitue souvent le bloc le plus complexe et le plus détaillé de tout le système. Elle doit donc être conçue avec le plus grand soin!
- Le diagramme ASM détaillé devrait facilement être converti en un chemin de contrôle, à condition que les règles susmentionnées soient suivies.

Diagramme détaillé ASM # 4



CU inputs = {Start, ARZ, ARG} = **Datapath** status outputs
CU outputs = {LdAB, Sel, LdCR, ClrCR, Done} =
= Control signals for **Datapath**





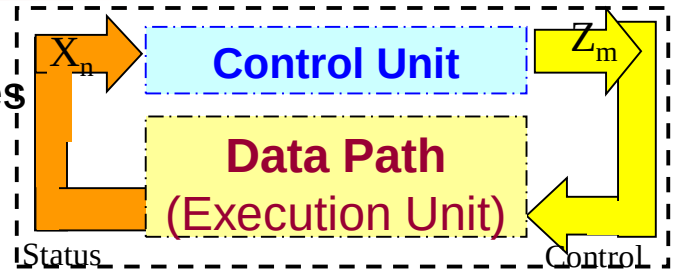
Conception de l'unité de contrôle (ASM # 5)

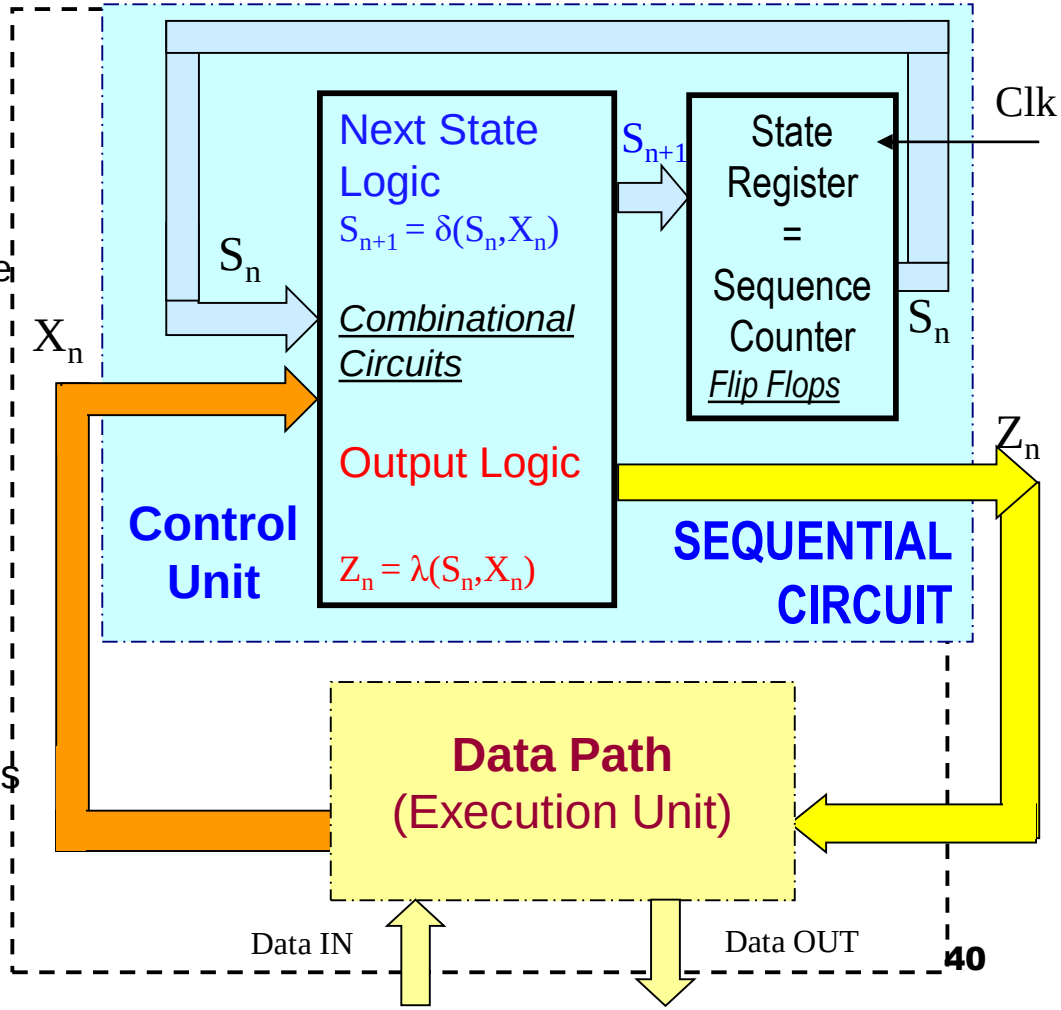
Il existe trois méthodes principales pour la mise en œuvre du registre d'état des circuits séquentiels utilisés comme circuits à logique de commande ASM:

1. La méthode d'un FF par état
   - Aussi appelé One-hot, nous verrons cette méthode dans de ce cours
   - Technique de conception la plus populaire, car elle est simple et facile à modéliser
2. La méthode d'état codé binaire
   - Utilise le codage d'état pour réduire le nombre de bascules D requises dans le registre d'état
   - La logique de contrôle est répartie sur plusieurs appareils numériques.
3. La méthode du compteur de séquence
   - Correspond à la nature cyclique de l'exécution d'une instruction sur un ordinateur
   - Le contrôle est simplifié en raison du modèle de transition d'état cyclique

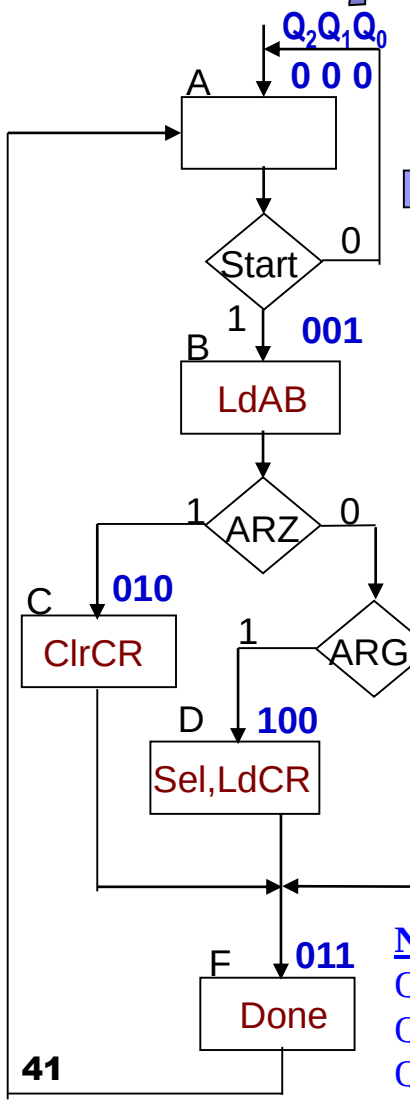
La méthode à base de PLD

- Fusionne tous les dispositifs logiques en une seule puce VLSI
- Peut contenir à la fois le chemin de données et la logique de contrôle!





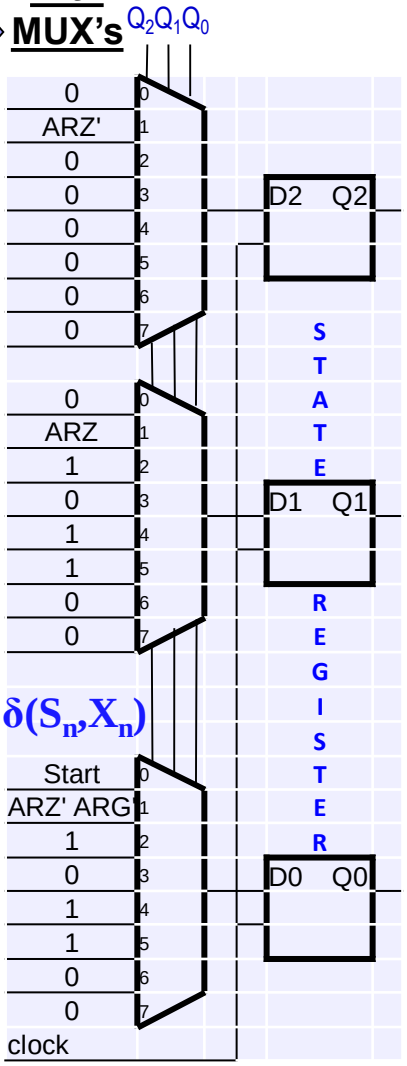
ASM #5.2



$\delta(S_n,X_n)$ $\lambda(S_n,X_n)$

State	Q2	Q1	Q0	Q2⁺	Q1⁺	Q0⁺	LdAB	LdCR	ClrCR	Sel	Done
A	0	0	0	0	0	Start	0	0	0	0	0
B	0	0	1	ARZ'	ARZ	ARZ' ARG'	1	0	0	0	0
C	0	1	0	0	1	1	0	0	1	0	0
F	0	1	1	0	0	0	0	1	0	1	0
D	1	0	0	0	1	1	0	1	0	0	0
E	1	0	1	0	1	1	0	0	0	0	1
	1	1	0	x	x	x	x	x	x	x	x
	1	1	1	x	x	x	x	x	x	x	x

with MUX's



with MEV = map entered variable

$Q0^+$

Q2 \ Q1Q0	00	01	11	10
0	0 Start	1 ARZ' ARG'	3 0	2 1
1	4 1	5 1	7 x	6 x

Output Logic $Z_n = \lambda(S_n)$

LdAB = B = Q2'Q1'Q0

LdCR = D+E =

= Q2'Q1Q0+Q2Q1'Q0'

ClrCR = C = Q2'Q1Q0'

Sel = D = Q2'Q1Q0

Done = F = Q2Q1Q0

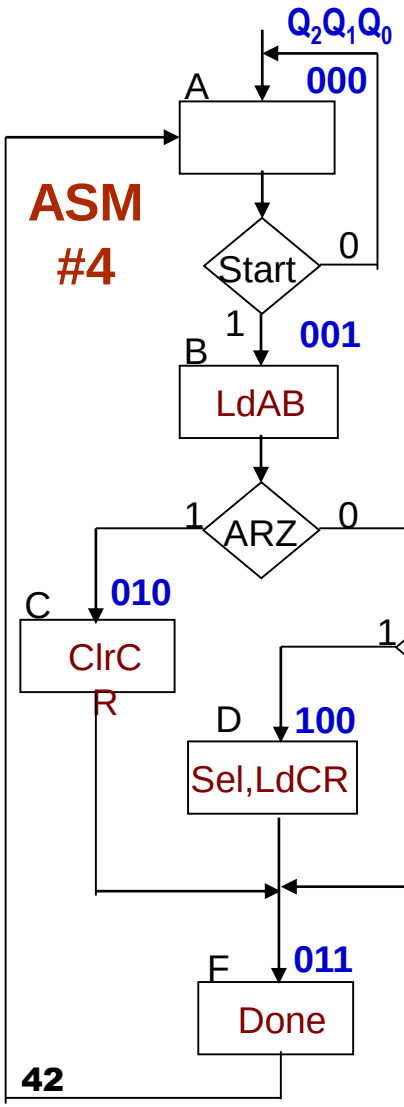
Next State Logic $S_{n+1} = \delta(S_n,X_n)$

$Q2^+ = B\cdot ARZ' = Q2'Q1'Q0 \cdot ARZ'$

$Q1^+ = B\cdot ARZ + C + D + E = Q1'Q0 \cdot ARZ + Q2 + Q1'Q0$

$Q0^+ = A\cdot start + B\cdot ARZ'\cdot ARG + C + D + E = Q0'\,start + Q1'Q0\cdot ARZ'\cdot ARG + Q2 + Q1Q0'$

ASM #5.3



ASM #4

State Register SC
= multi-function register $Q_2Q_1Q_0$ with
- parallel load (LdSC),
- count up (SC++),
- clear (ClrSC)

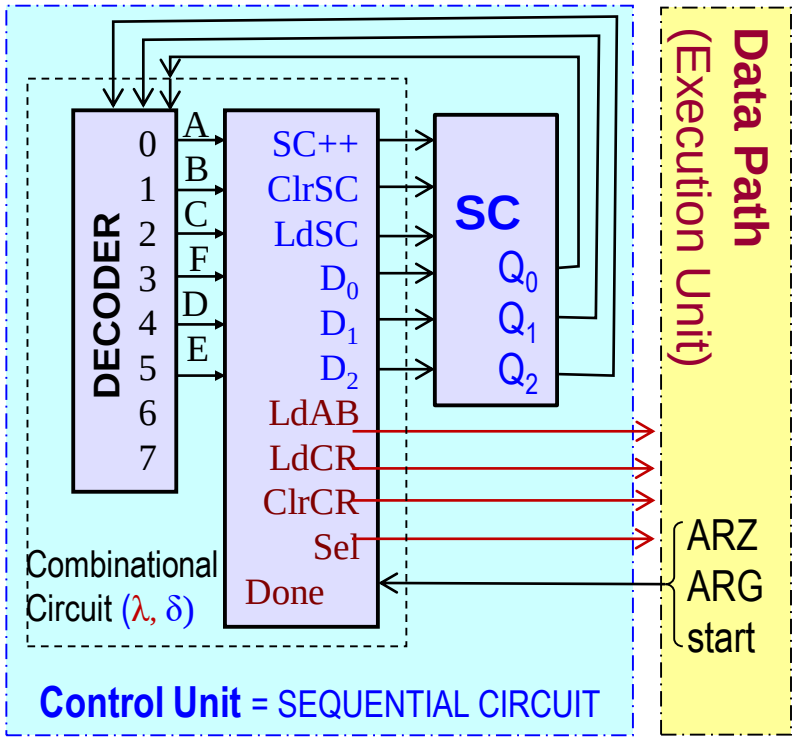
ASM #2 / RTL			δ				λ				
	$Z^n = \lambda$	$S^{n+1} = \delta$	SC++	ClrSC	LdSC	SC^{n+1} $D_2D_1D_0$	LdAB	LdCR	ClrCR	Sel	Done
A Start:		SC←B	1								
B:	AR← DB-A, BR← DB-B						1				
B ARZ:		SC←C	1								
B·ARZ'·ARG:		SC←D			1	100					
B·ARZ'ARG':		SC←E			1	101					
C:	CR←0	SC←F	1						1		
D:	CR←BR*2	SC←F			1	011		1		1	
E:	CR←AR/2	SC←F			1	011		1			
F:		SC←A		1							1

Next State Logic $S_{n+1} = \delta(S_n, X_n)$

$SC{+}{+} = A{\cdot}start + B{\cdot}ARZ + C$
$ClrSC = F$
$LdSC = B{\cdot}ARZ'{\cdot}ARG + B{\cdot}ARZ'ARG' + D + E$
$D_0 = B{\cdot}ARZ'ARG' + D + E$
$D_1 = D + E$
$D_2 = B{\cdot}ARZ'{\cdot}ARG + B{\cdot}ARZ'ARG'$

Output Logic $Z_n = \lambda(S_n)$

$LdAB = B$
$LdCR = D + E$
$ClrCR = C$
$Sel = D$
$Done = F$



Règles pour une conception sûre

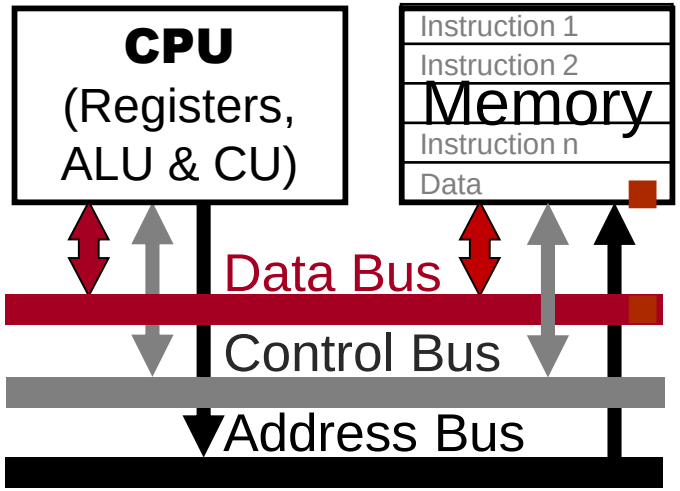
1. Gardez la conception simple et modulaire:
   - Utilisez une procédure itérative, si nécessaire, pour affiner le processus;
   - Implémentez un module à la fois, tout en testant chaque module;
2. Développer une bonne documentation lors de la conception;
3. Attention au horloge biaisé;
   - Utilisez des longueurs de chemin similaires pour toutes les bascules;
   - Ne pas utiliser des horloges commandés par des portes!
   - Utilisez toutes les bascules à déclenchement positif ou toutes les bascules à déclenchement négatif;
4. Méfiez-vous des entrées asynchrones du circuit;
   - Évitez-les autant que possible!
   - Synchronisez ceux qui ne peuvent pas être évités;
   - Utilisez des commutateurs anti-rebond pour fournir des signaux d'entrée nets;
5. Méfiez-vous du bruit sur les lignes d'alimentation et de signal;
6. Éviter les dépendances sur les délais minimaux des portes logiques;
7. Initialisez toutes les bascules aux valeurs connues au début.

ORDINATEUR DE BASE CYCLE D'INSTRUCTION

Spécifications RTL ASM et conception générale

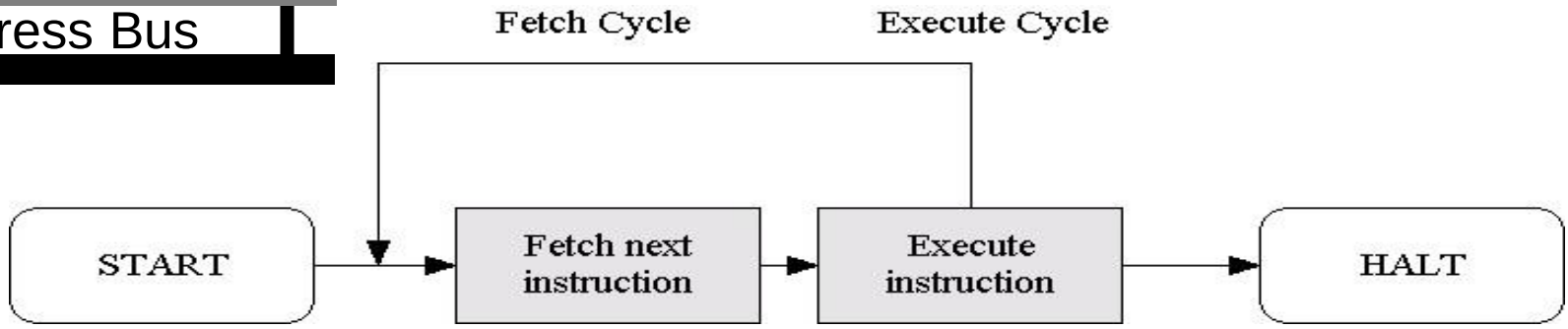
Cycle d'instruction de base

- Un programme résidant dans la mémoire de l'ordinateur consiste en une séquence d'instructions. Chaque instruction est exécutée en plusieurs étapes:



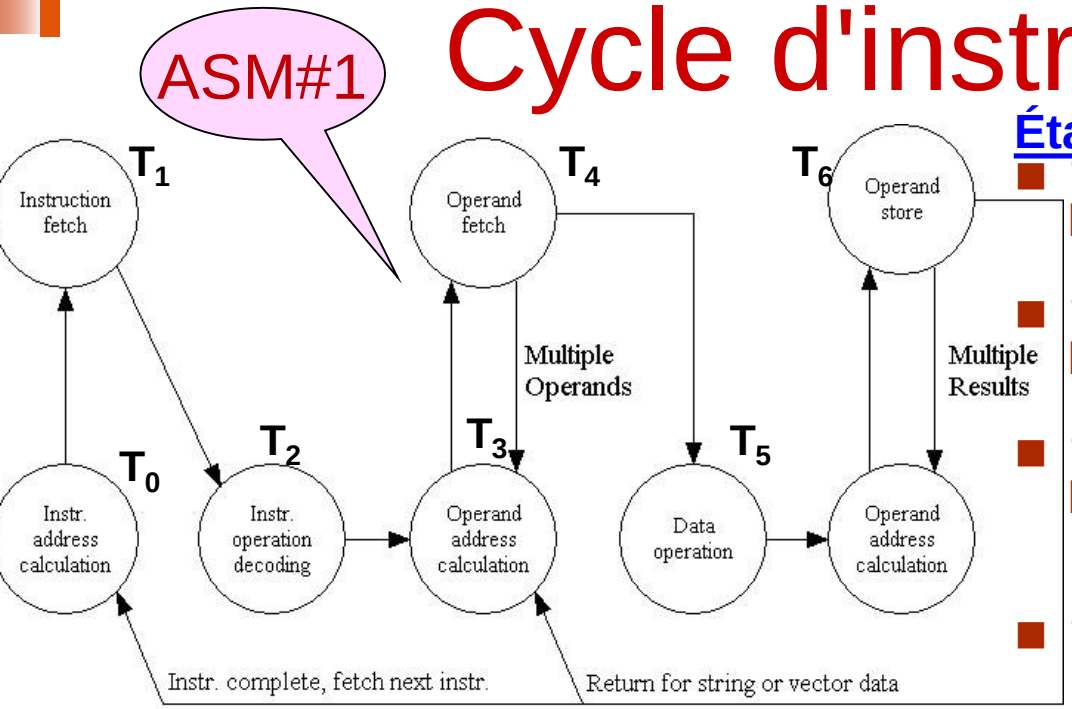
1. Récupère une instruction de la mémoire.
2. Décoder l'instruction.
3. Lisez l'adresse effective dans la mémoire si l'instruction a une adresse indirecte.
4. Exécutez l'instruction.

À la fin de l'étape 4, la commande revient à l'étape 1 pour recommencer le cycle pour l'instruction suivante. Le processus est répété indéfiniment sauf si une instruction HALT est rencontrée.



- L'ordinateur effectue le cycle d'instruction pour toujours! (ou au moins jusqu'à ce qu'il soit éteint, fait face à une erreur ou est invité à arrêter)

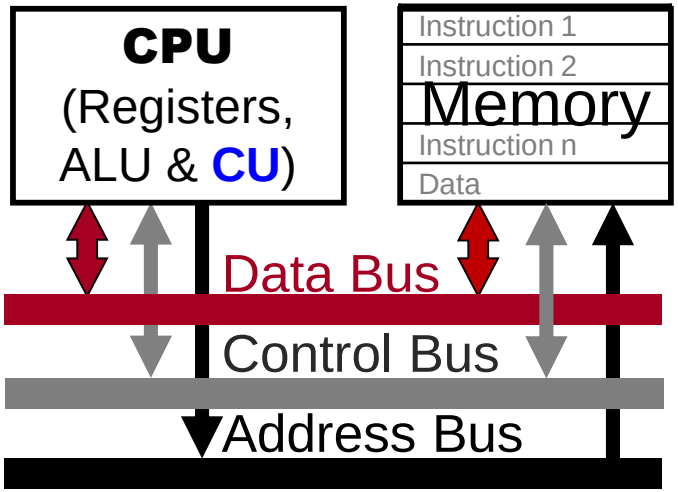
Cycle d'instruction détaillé



États de l'unité de contrôle: T_0, T_1, T_2, …

- T_0 - Calcul d'adresse d'instruction
  - Détermine l'adresse de la prochaine instruction à exécuter (PC)
- T_1 - Récupération d'instruction
  - Lit les instructions de l'emplacement mémoire dans le processeur - IR
- T_2 - Décodage de l'opération d'instruction
  - Analyse l'instruction pour déterminer le type d'opération à effectuer et le ou les opérandes à utiliser
- T_3 - Calcul d'adresse d'opérande
  - Détermine l'adresse de l'opérande (si nécessaire) - RA
- T_4 - Exécution de l'instruction Opérande extraire
  - Récupère l'opérande de la mémoire ou le lit des E/S -> AC
- T_5 – Exécution de l'instruction Opération de données
  - Effectue l'opération indiquée dans l'instruction (ALU)
- T_6 - Exécution de l'instruction d'opérande de stockage
  - Ecrivez les résultats en mémoire ou envoyez vers les E/S

L'unité de contrôle (CU) qui implémente ce qui précède



Les étapes ont une structure semblable à l'exemple ASM#5.3 de la section précédente de l'ASM.

Ordinateur de base

uOttawa

1. **Fetch** instruction from memory location specified by **PC** to Instruction Register (**IR**)

T_0: AR ← [PC] - Transfer contents of PC to Memory Address Register

T_1: IR <- [M(AR)] - Fetch instruction = contents of addressed memory location [M(AR)] is transferred to the IR

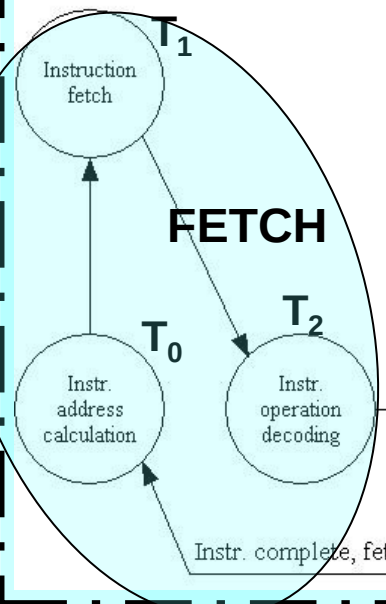
PC<-PC+1 - PC incremented to point to the next instruction

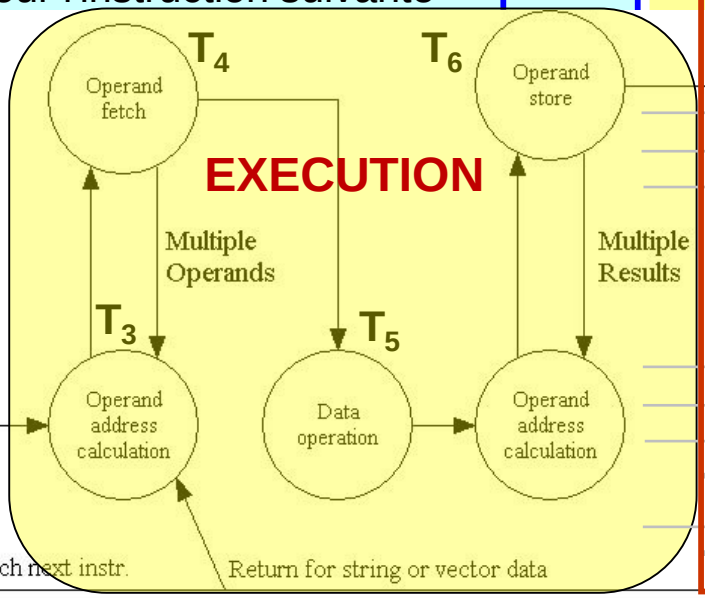
T_2 Instruction decoded by **CU**

2. **Execute** instruction held in IR

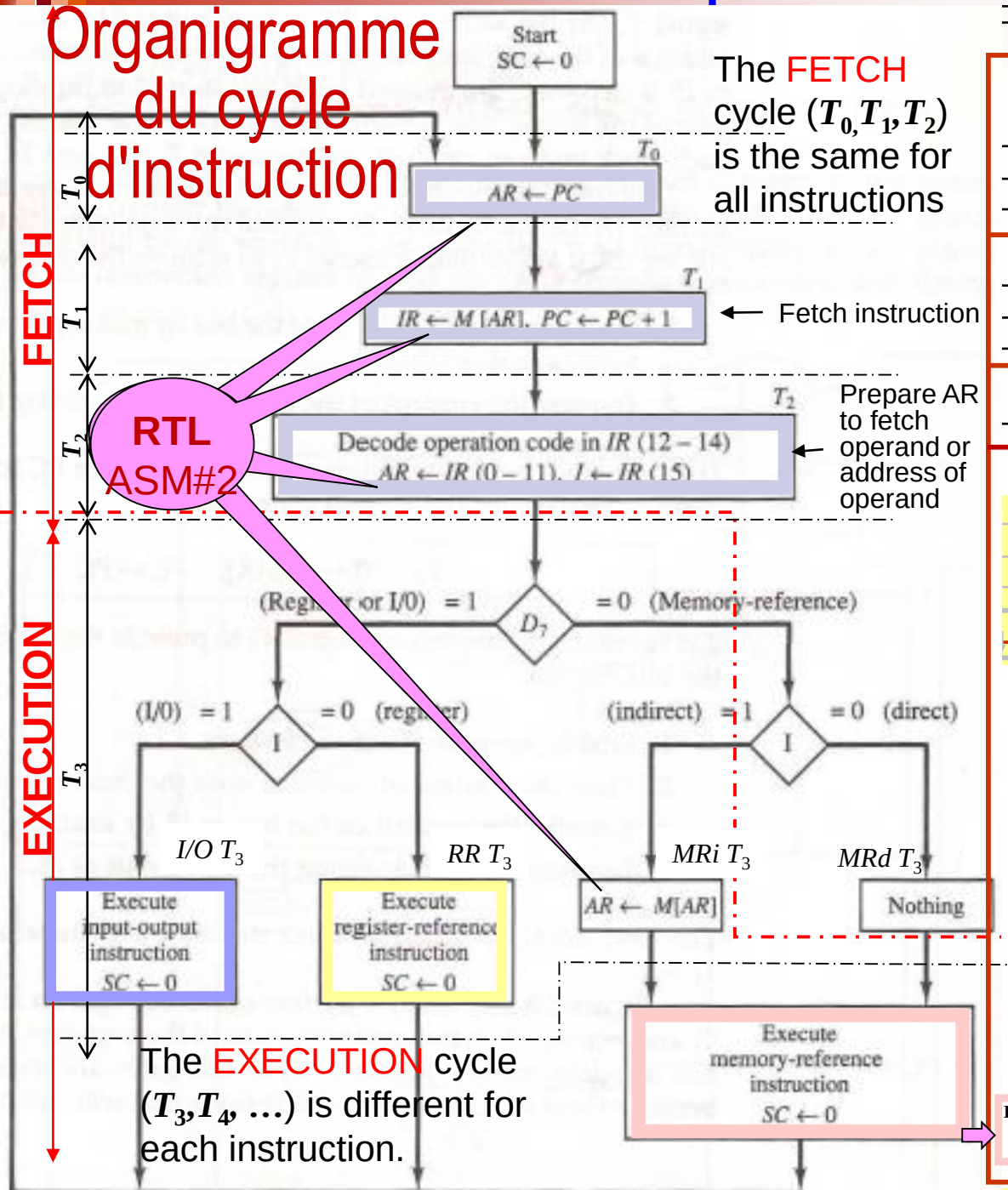
T3 …

Processus répété pour l'instruction suivante





Organigramme du cycle d'instruction



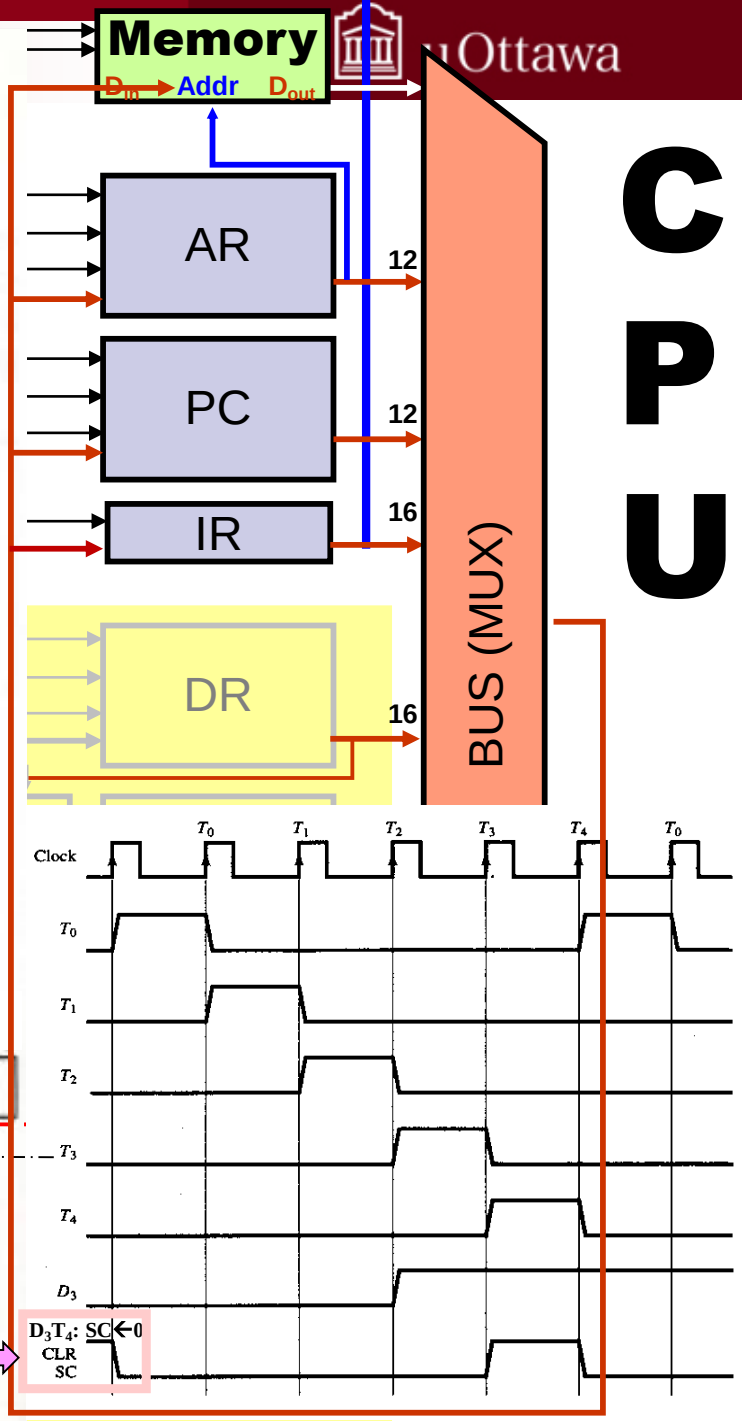
RTL ASM#2

The FETCH cycle (T_0, T_1, T_2) is the same for all instructions

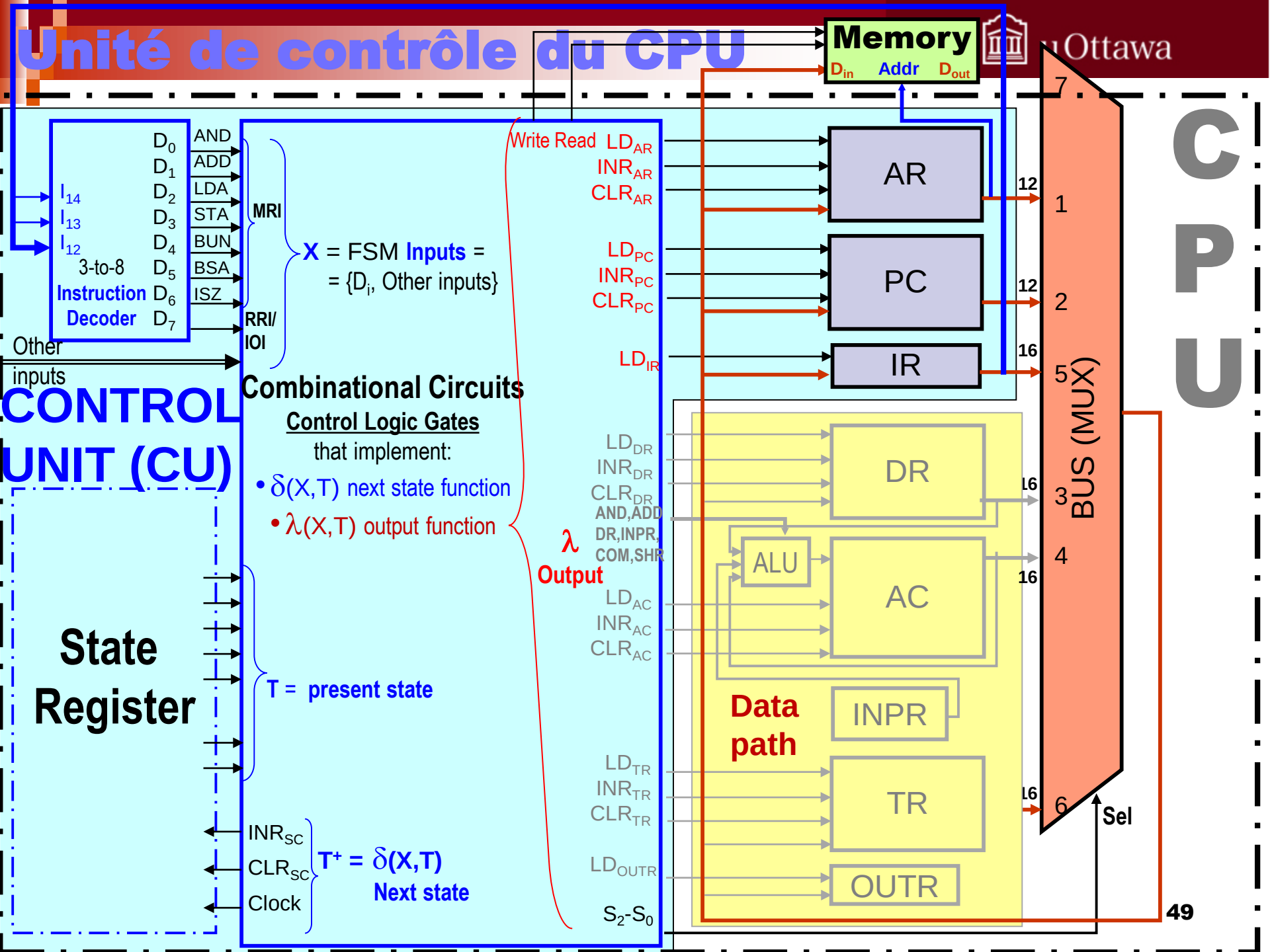
← Fetch instruction

Prepare AR to fetch operand or address of operand

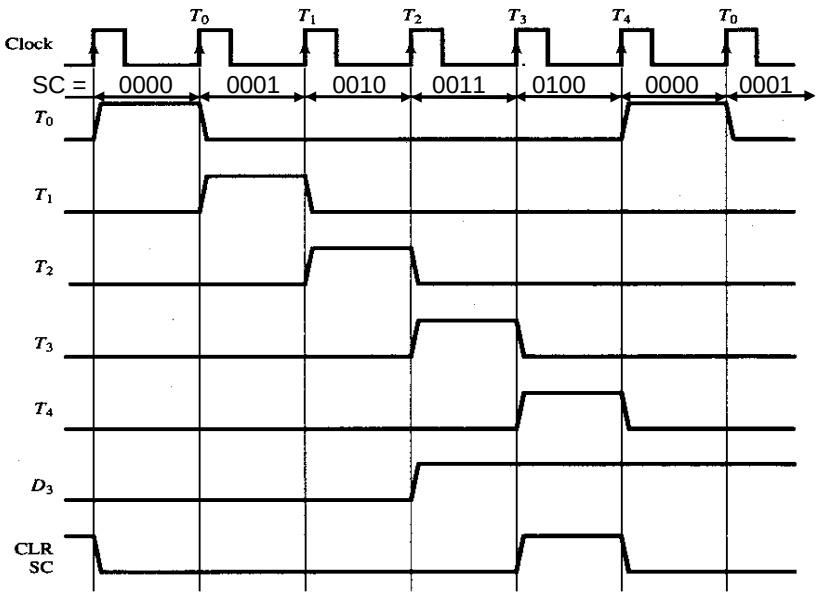
The EXECUTION cycle (T_3, T_4, …) is different for each instruction.



Unité de contrôle du CPU



Unité de contrôle du CPU
(registre d'état + δ)



ASM#5.3

Instruction register (*IR*)

15 | 14 13 12 | 11–0

3 × 8 decoder

7 6 5 4 3 2 1 0

ISZ BSA BUN STA LDA ADD AND **(MRI)**

RRI & IOI

Other inputs X_n

Control logic gates

Control outputs

Z_n

Combinational Circuits

λ(X,T) output function

State Register

4 × 16 decoder

4-bit sequence counter (SC)

Increment (INR)

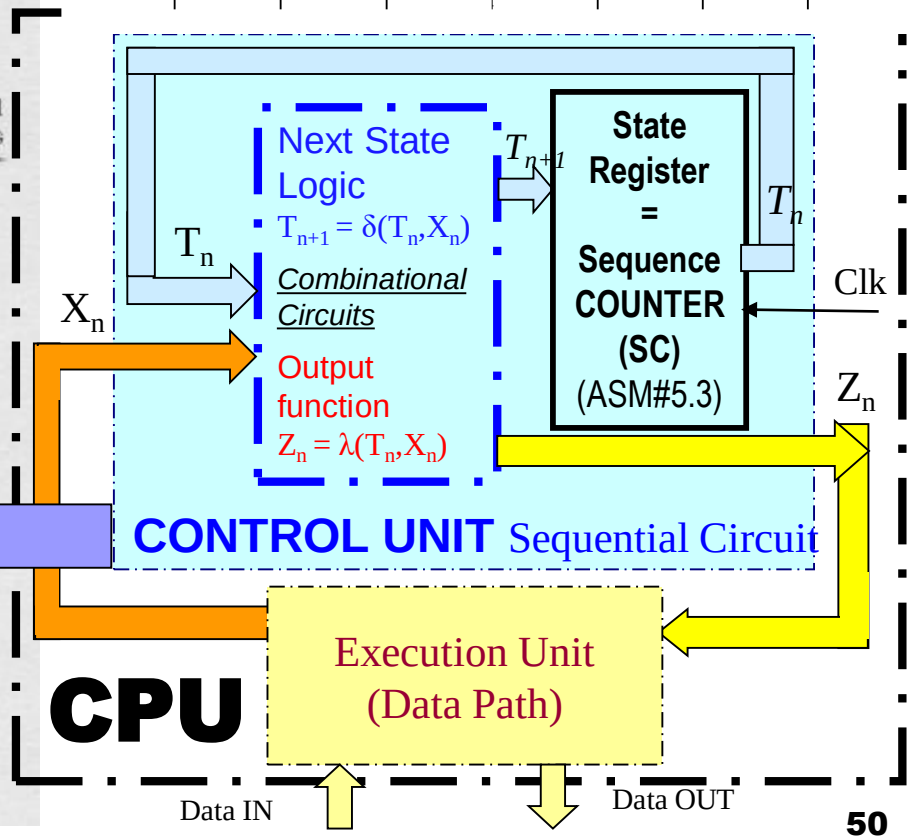
Clear (CLR)

Clock

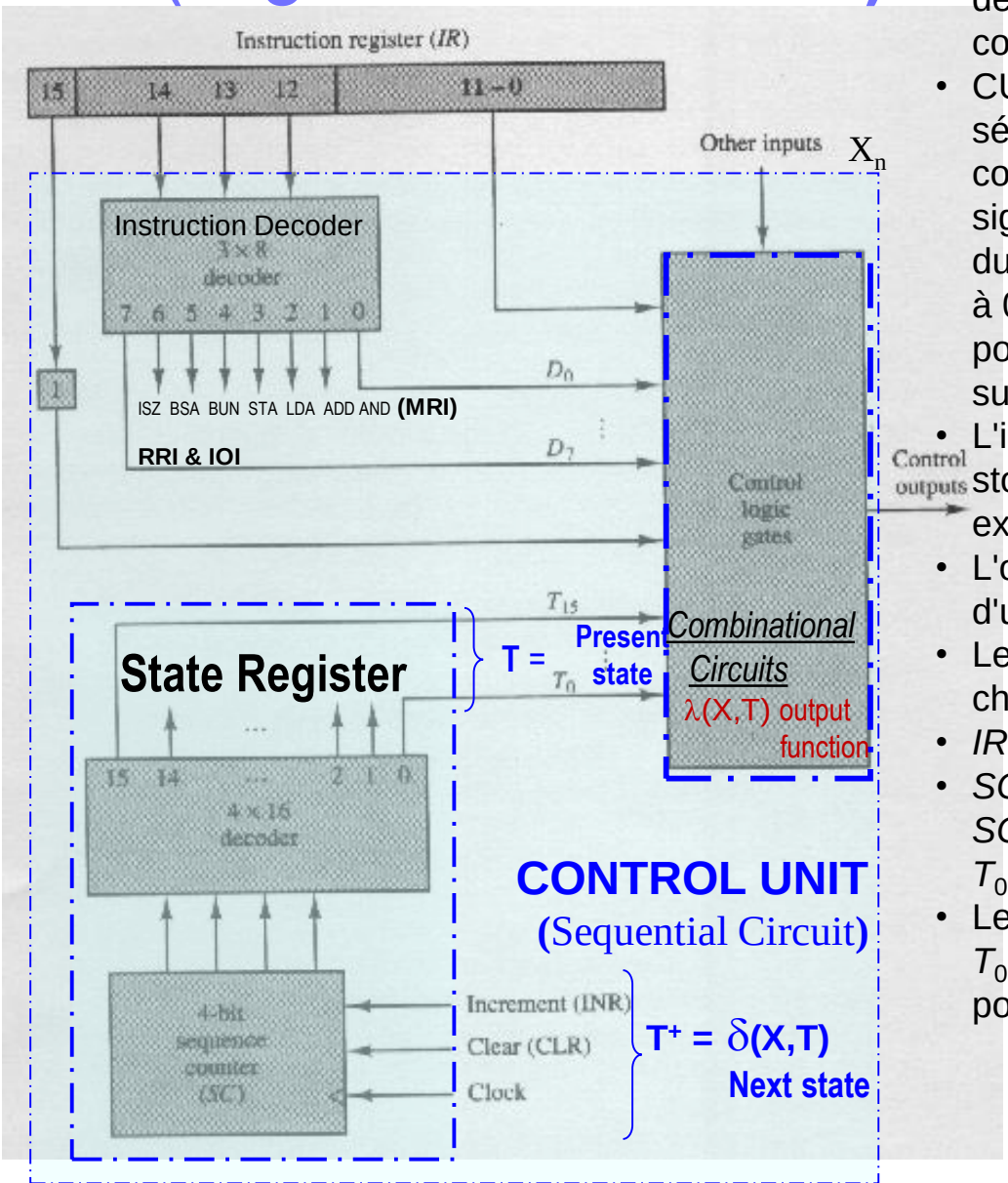
T = T_0 … T_{15} Present state

$T^+ = \delta(X,T)$ Next state

CONTROL UNIT
(Sequential Circuit)

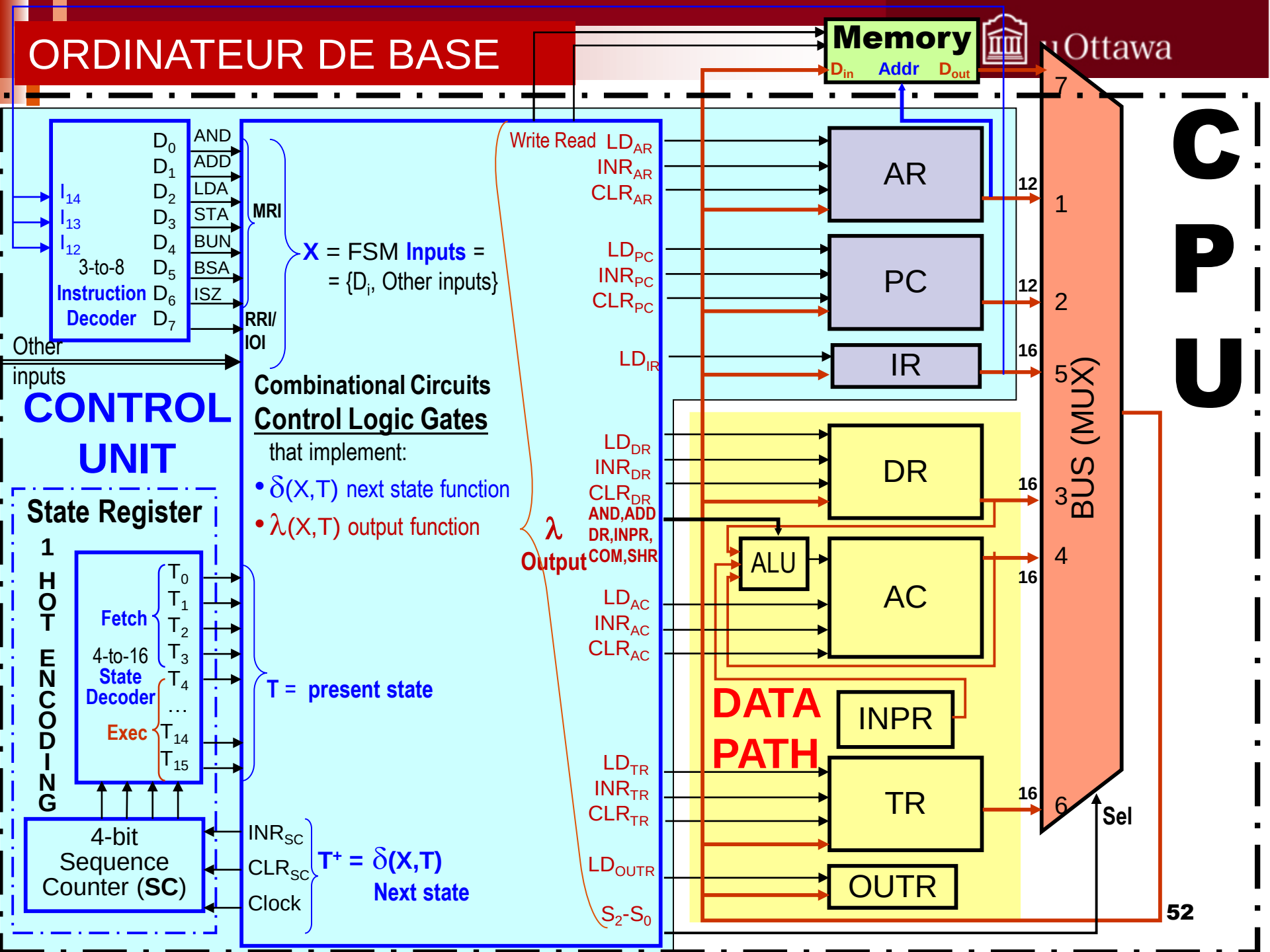


Unité de contrôle du CPU (registre d'état + δ)



- L'unité de commande (CU) est un circuit séquentiel programmable dont les fonctions de transition(δ) et de sortie (λ) changent en fonction de l'instruction en cours I_{14}-I_0.
- CU consiste en un registre d'état (compteur de séquence SC + *décodeur* 4x16) et de circuits combinatoires qui calculent la fonction de sortie (λ - signaux de commande des multiplexeurs et registres du système) et la fonction de transition (δ - CLR met à 0 SC à la fin de l'exécution de chaque instruction pour ramener SC à l'état initialT_0, pour l'instruction suivante).
- L'instruction en cours est lue de la mémoire et est stockée dans le registre d'instruction IR pendant son exécution.
- L'opcode $IR_{14}IR_{13}IR_{12}$ à 3 bits est décodé à l'aide d'un décodeur 3 x 8 (*décodeur d'instruction*);
- Les sorties du décodeur d'instructions sont D_0 - D_7, chacune correspondant à une opération.
- IR_{15} est transféré sur une bascule *I*.
- *SC* peut compter en binaire de 0 à 15. *Les sorties de SC sont décodées en 16 signaux de synchronisation* T_0-T_{15}
- Les 12 bits les moins significatifs de IR, *I*, D_0-D_7, et T_0 - T_{15}, sont tous transmis en tant qu'entrées aux portes de la logique de commande.

ORDINATEUR DE BASE

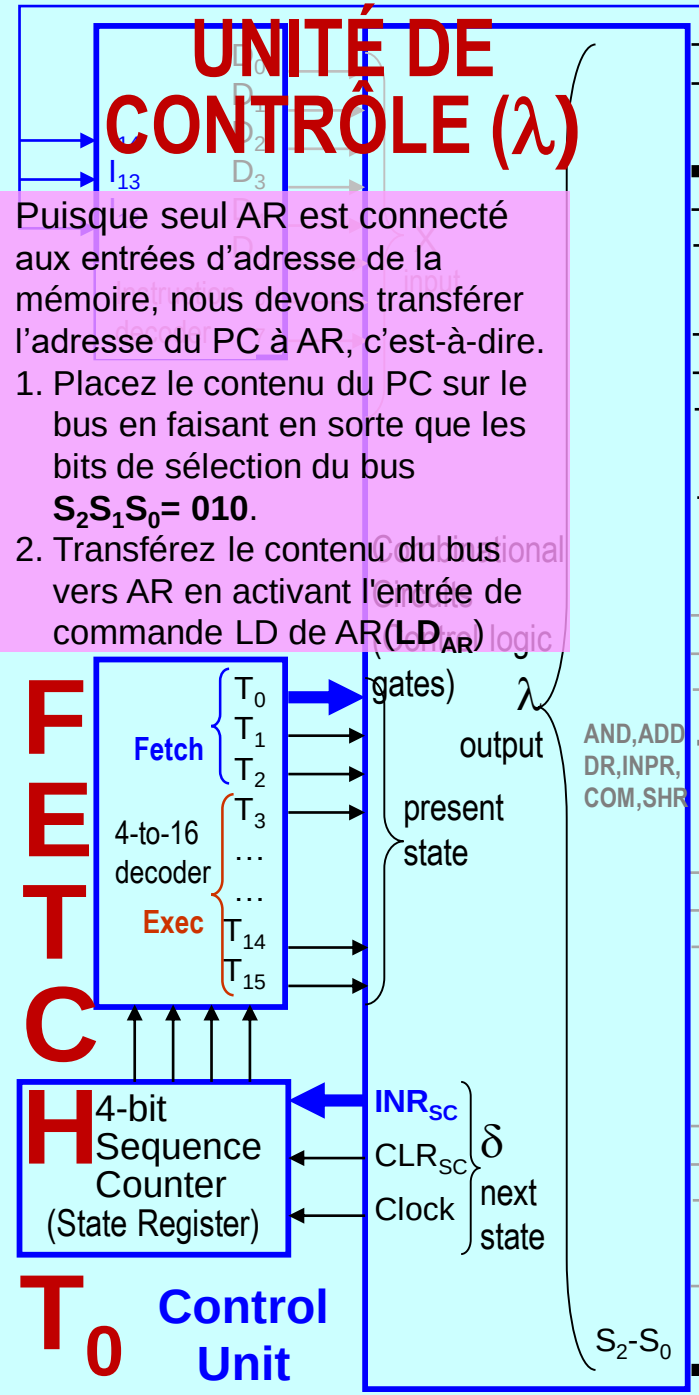


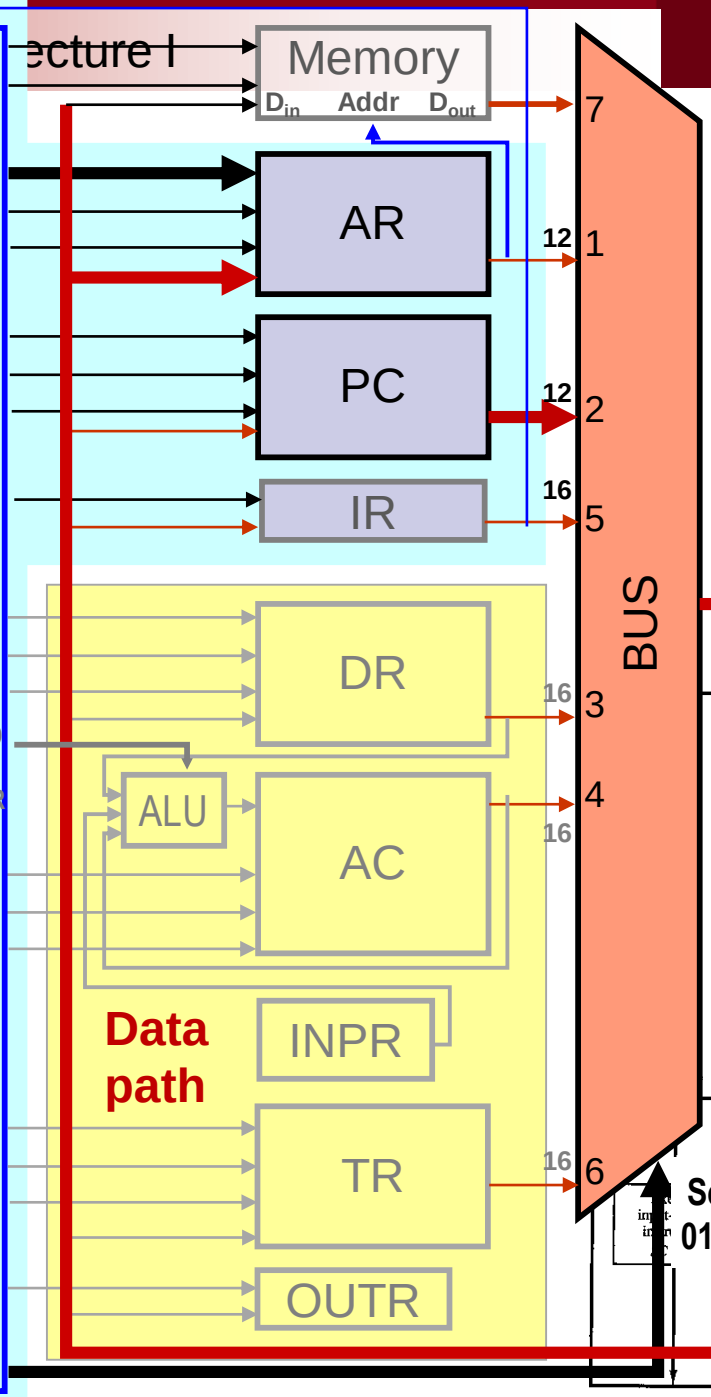
Spécifications détaillées de l'instruction RÉCUPÉRATION (FETCH) en RTL ASM (ASM # 2) et avec ASM détaillée (ASM # 4)

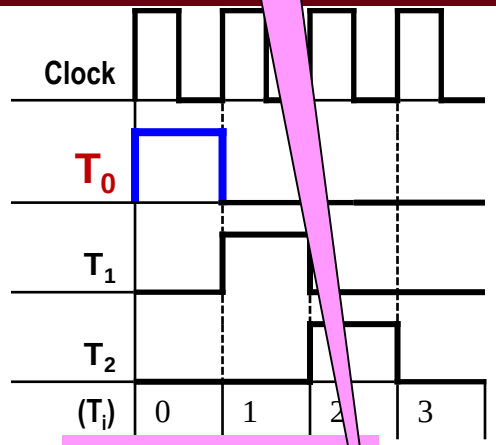
UNITÉ DE CONTRÔLE (λ)

Puisque seul AR est connecté aux entrées d'adresse de la mémoire, nous devons transférer l'adresse du PC à AR, c'est-à-dire.

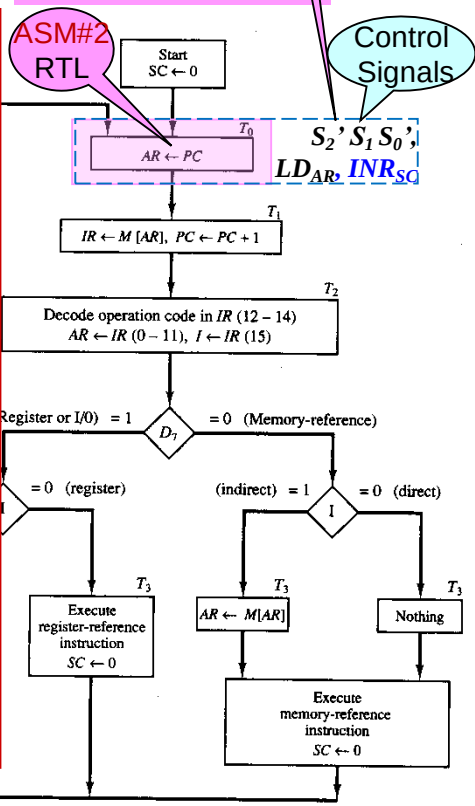
1. Placez le contenu du PC sur le bus en faisant en sorte que les bits de sélection du bus **$S_2S_1S_0$= 010**.
2. Transférez le contenu du bus vers AR en activant l'entrée de commande LD de AR(**LD_{AR}**)







T_0: AR <- PC



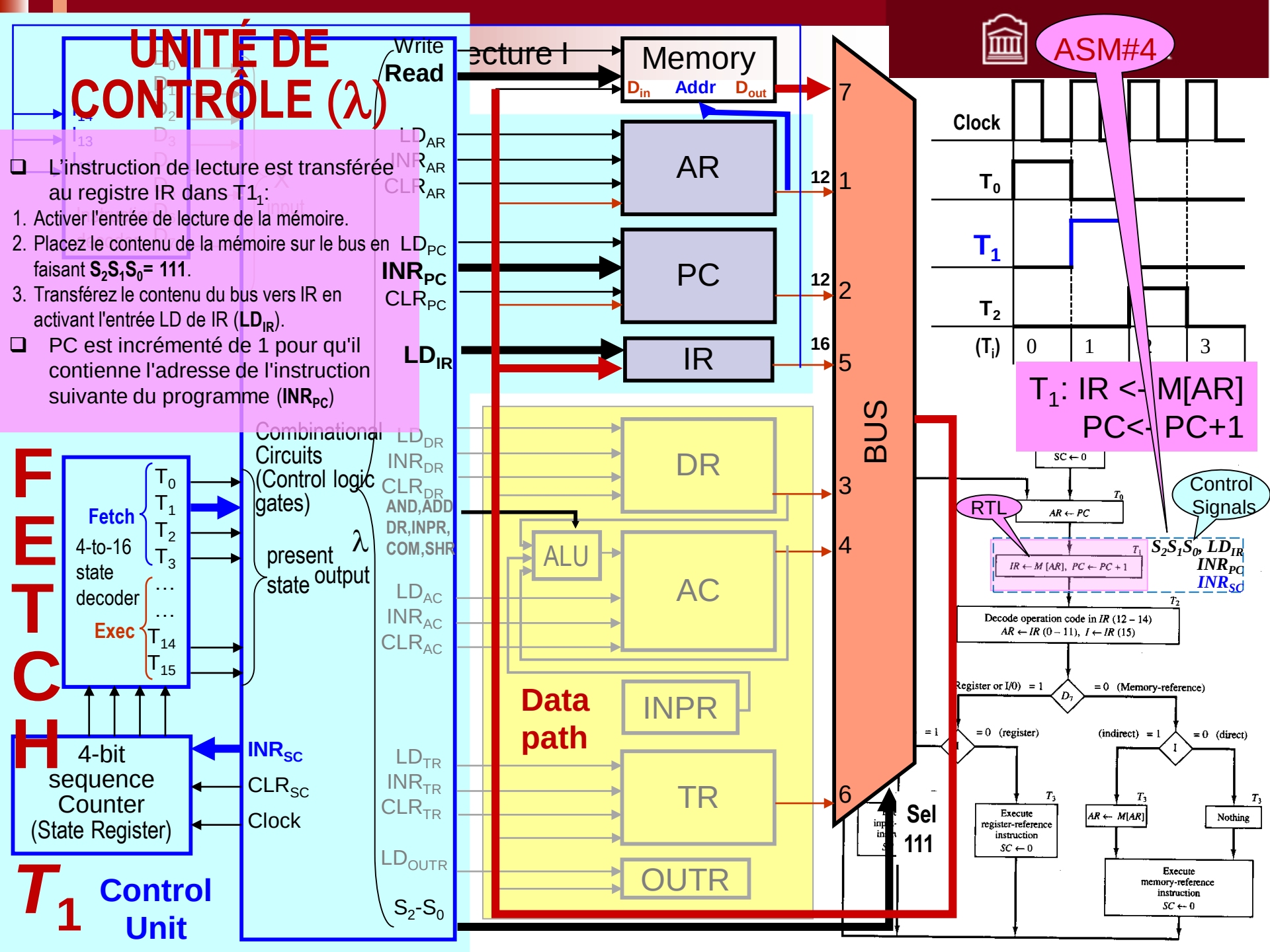
Sel 010

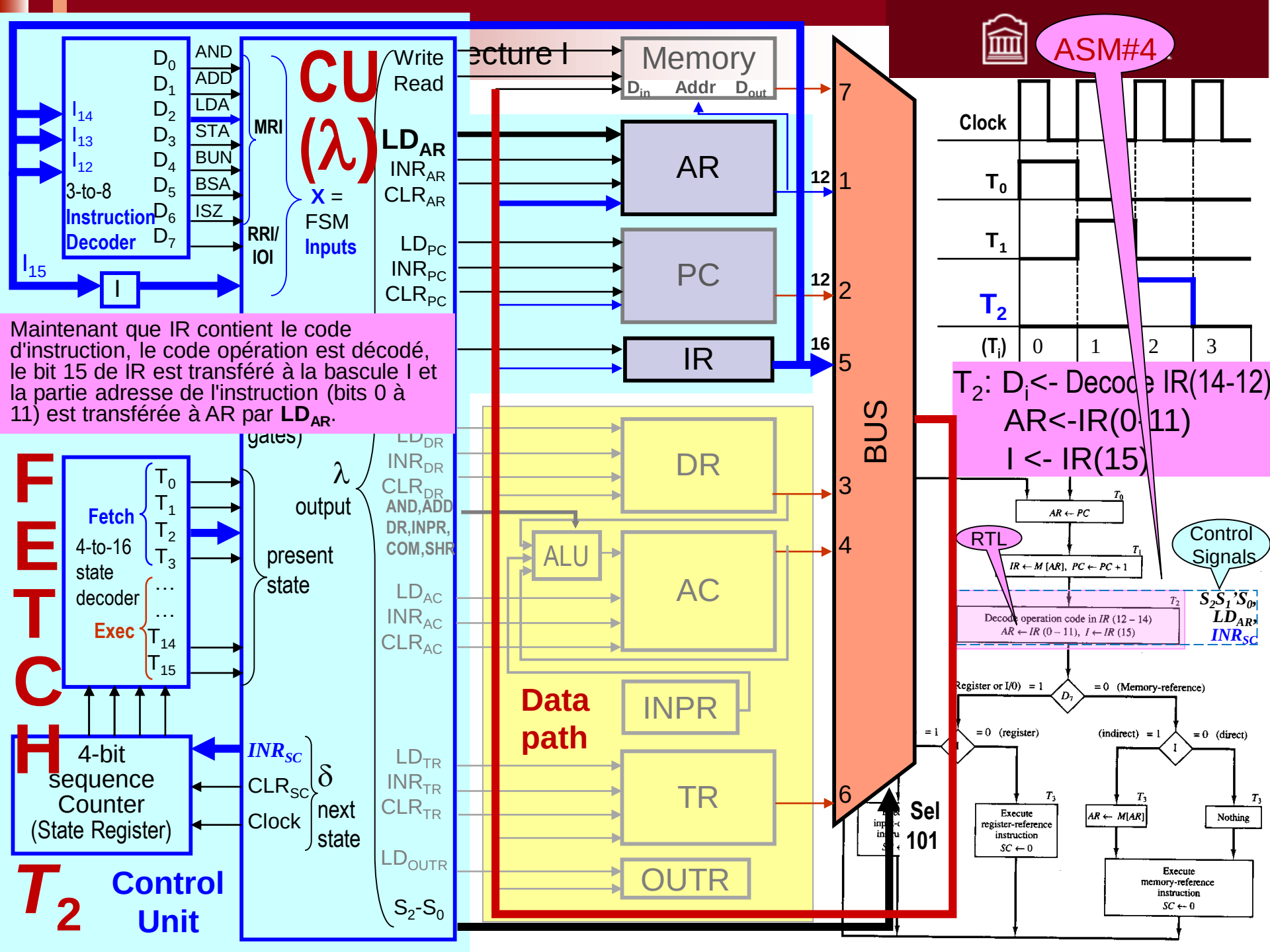
UNITÉ DE CONTRÔLE (λ)

- L'instruction de lecture est transférée au registre IR dans $T1_1$:

1. Activer l'entrée de lecture de la mémoire.
2. Placez le contenu de la mémoire sur le bus en faisant $S_2S_1S_0$= 111.
3. Transférez le contenu du bus vers IR en activant l'entrée LD de IR (LD_{IR}).

- PC est incrémenté de 1 pour qu'il contienne l'adresse de l'instruction suivante du programme (INR_{PC})




ecture I
Memory
Din Addr Dout
CU (λ)
X = FSM Inputs
Write
Read
LDAR
INRAR
CLRAR
LDPC
INRPC
CLRPC
AR
PC
IR
I
I14
I13
I12
I15
3-to-8 Instruction Decoder
D0 AND
D1 ADD
D2 LDA
D3 STA
D4 BUN
D5 BSA
D6 ISZ
D7
MRI
RRI/ IOI
Maintenant que IR contient le code d'instruction, le code opération est décodé, le bit 15 de IR est transféré à la bascule I et la partie adresse de l'instruction (bits 0 à 11) est transférée à AR par LDAR.
gates)
LDDR
INRDR
CLRDR
DR
λ output
AND,ADD DR,INPR, COM,SHR
ALU
AC
LDAC
INRAC
CLRAC
Data path
INPR
LDTR
INRTR
CLRTR
TR
LDOUTR
OUTR
S2-S0
BUS
7
12
1
12
2
16
5
3
4
6
Sel 101
FETCH
Fetch
4-to-16 state decoder
T0
T1
T2
T3
Exec
T14
T15
present state
4-bit sequence Counter (State Register)
INRSC
CLRSC
Clock
δ next state
T2 Control Unit
Clock
T0
T1
T2
(Ti) 0 1 2 3
T2: Di<- Decode IR(14-12)
AR<-IR(0-11)
I <- IR(15)
RTL
Control Signals
AR ← PC
IR ← M [AR], PC ← PC + 1
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
S2S1'S0
LDAR
INRSC
Register or I/O) = 1
D7
= 0 (Memory-reference)
= 1
I
= 0 (register)
(indirect) = 1
= 0 (direct)
Execute register-reference instruction SC ← 0
AR ← M[AR]
Nothing
Execute memory-reference instruction SC ← 0

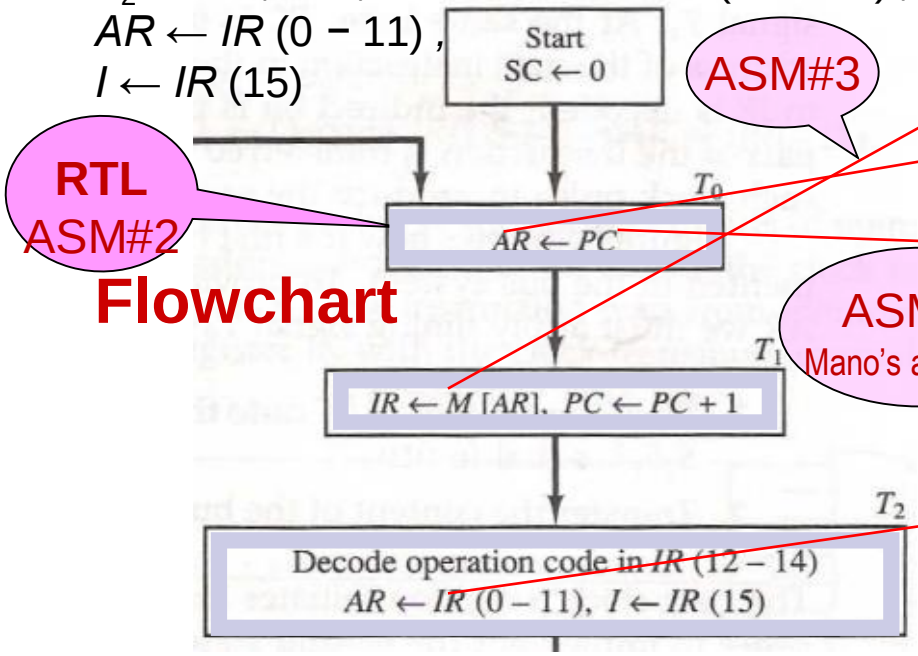
Fetch&Decode

- Lorsqu'un programme est exécuté, le compteur de programme PC reçoit initialement l'adresse de la première instruction du programme et le compteur d'état SC est effacé, ce qui donne T_0=1
- Après chaque impulsion d'horloge, SC est automatiquement incrémenté, de sorte que les signaux de synchronisation passent par les états T_0, T_1, T_2, … et que l'ASM avance
- Les micro-opérations des phases d'extraction et de décodage peuvent être spécifiées à l'aide des instructions RTL suivantes:

T_0 : $AR \leftarrow PC$

T_1 : $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$

T_2 : $D0, \ldots, D7 \leftarrow$ Decode $IR(12-14)$, $AR \leftarrow IR(0-11)$, $I \leftarrow IR(15)$

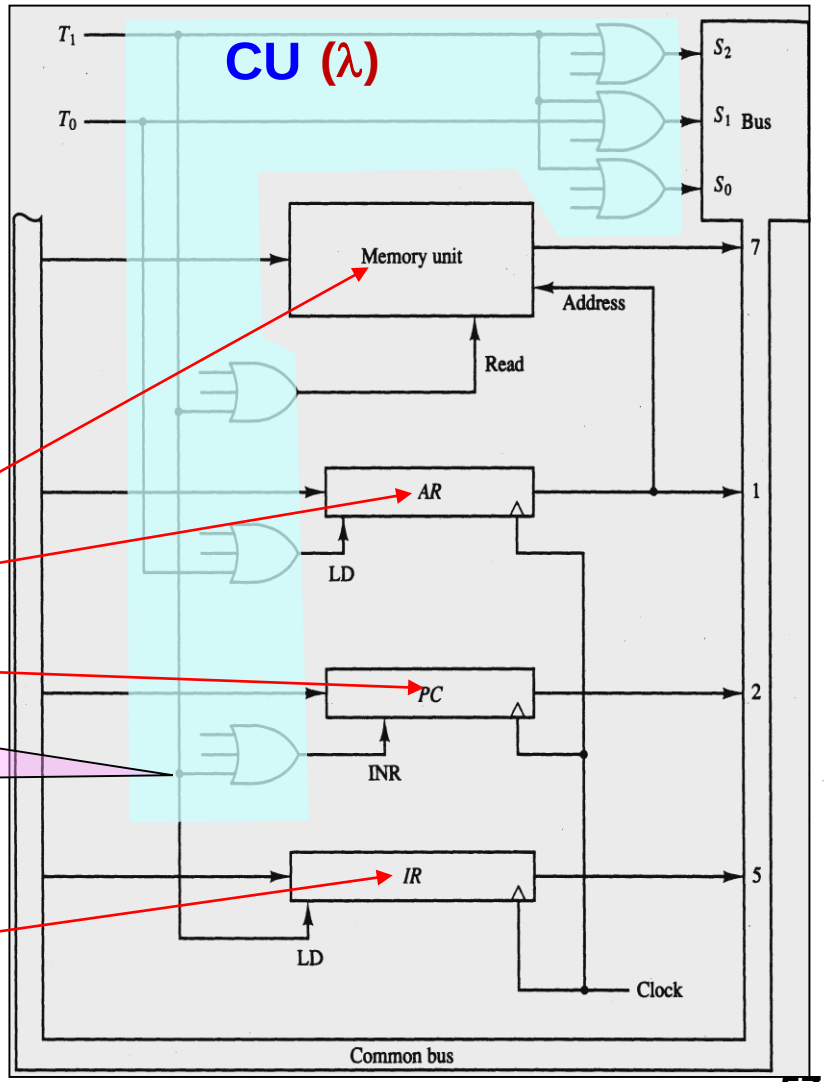


RTL ASM#2

Flowchart

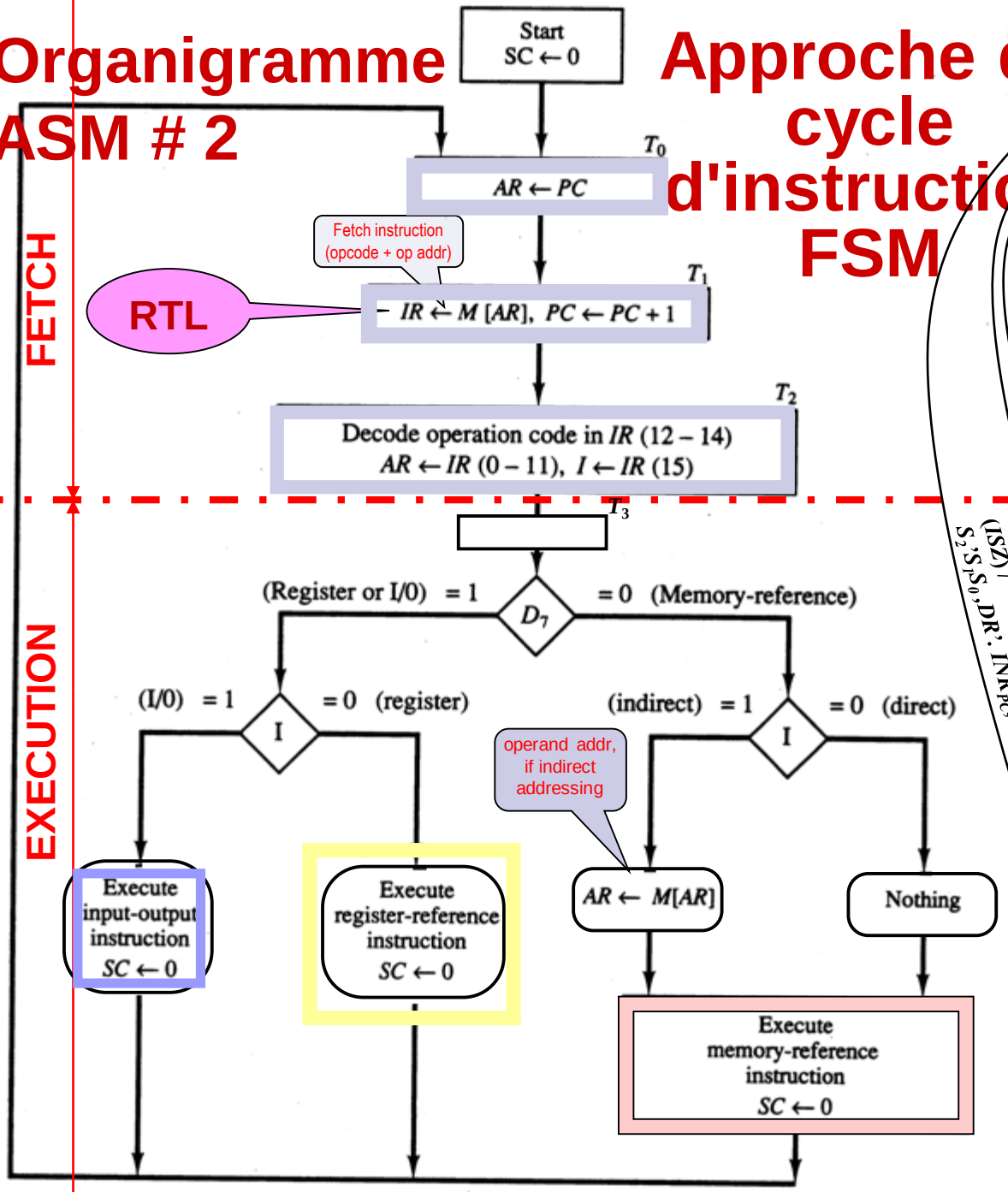
ASM#3

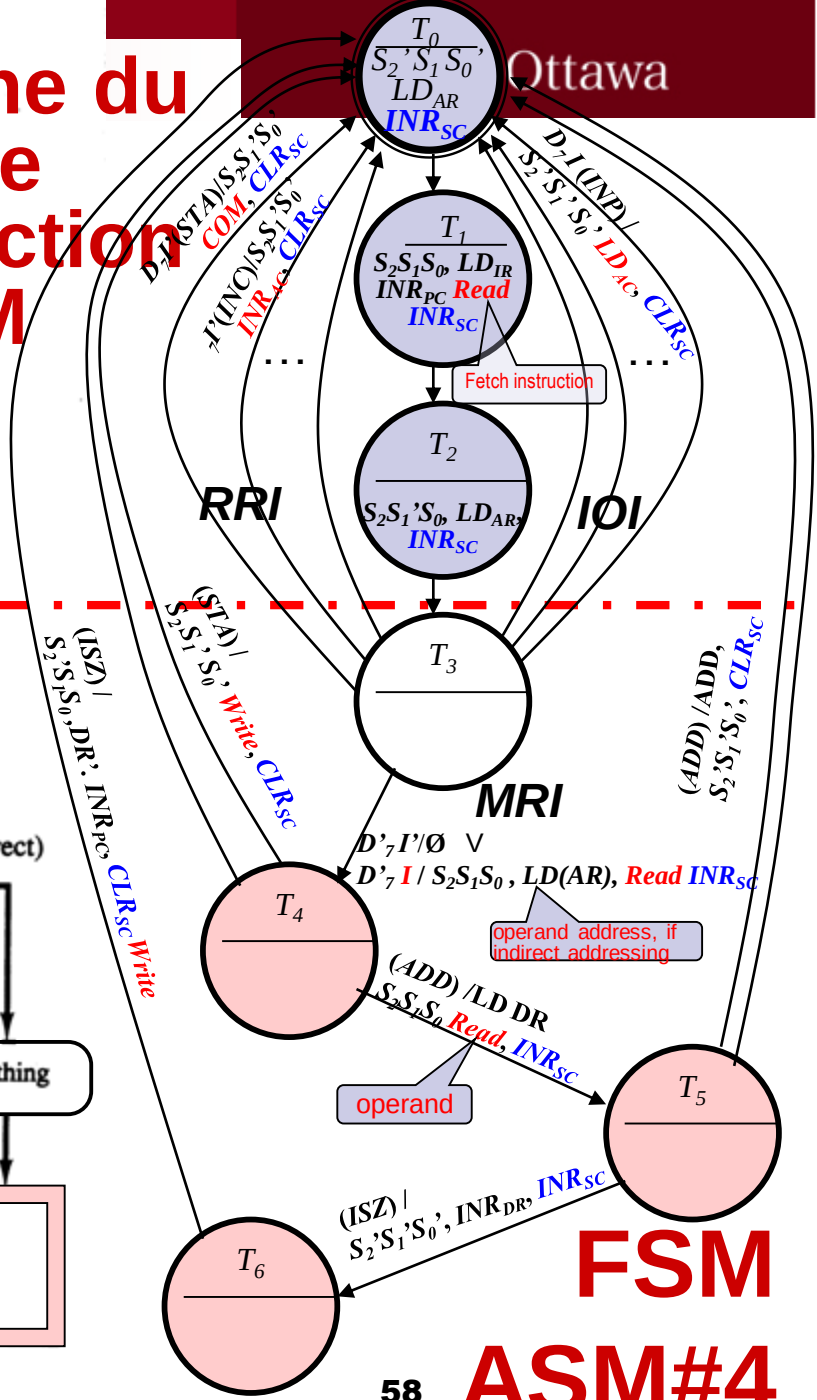
ASM#5 Mano's approach

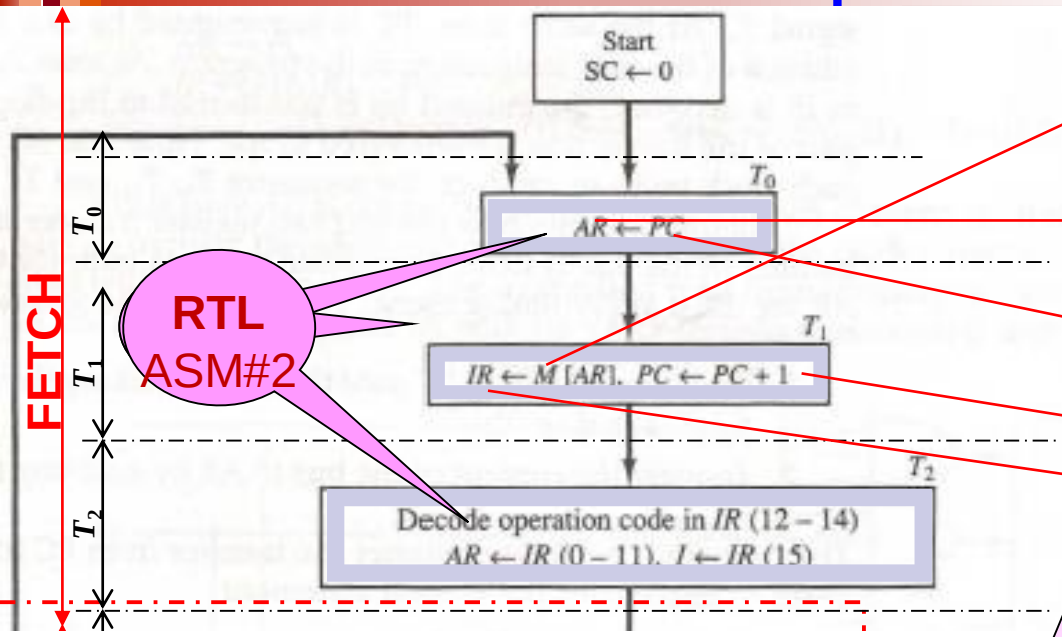


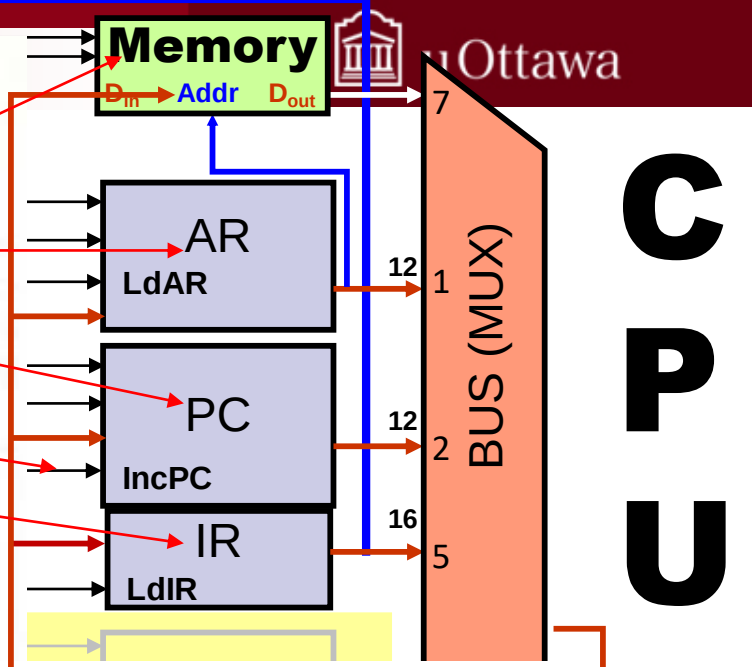
Organigramme ASM # 2

Approche du cycle d'instruction FSM





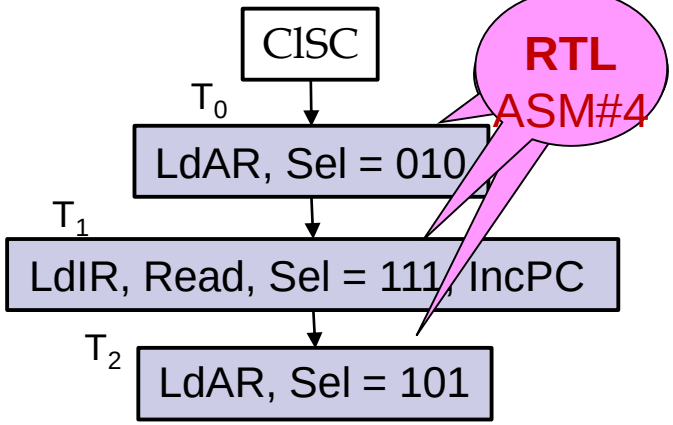




CPU

Instruction FETCH et signaux de contrôle de synchronisation

ASM#3



ASM#5

CU (λ)

$LdAR = T_0 + T_2 + \ldots$

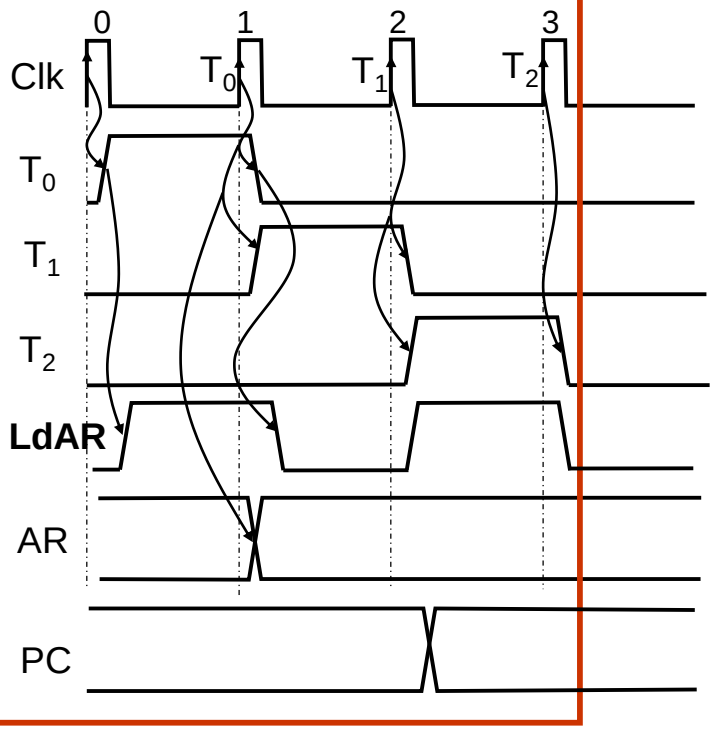
$LdIR = T_1 + \ldots$

$Read = T_1 + \ldots$

$Sel_2 = T_1 + T_2 + \ldots$

$Sel_1 = T_0 + T_1 \ldots$

$Sel_0 = T_1 + T_2 + \ldots$



Impossible de terminer le projet faute de spécifications sur l'exécution des instructions!

Toutes les instructions détaillent les spécifications de EXECUTION en RTL ASM (ASM # 2)

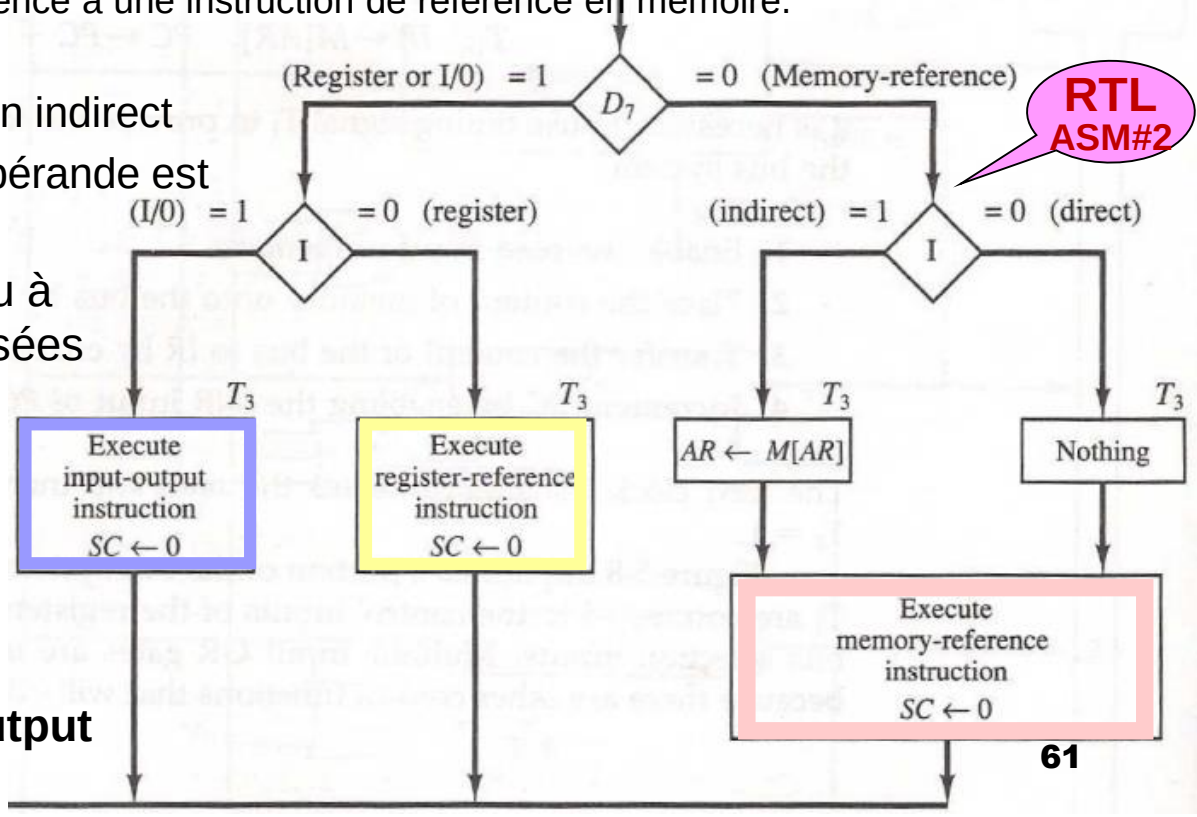
et détaillé ASM (ASM # 4)

Déterminer le type d'instruction

- La récupération et le décodage des instructions ont lieu aux instants T_0 à T_2.
- Pendant T_3, CU détermine le type d'instruction à lire dans la mémoire.
- Le segment suivant de l'organigramme ASM pour le cycle d'instruction présente une configuration initiale du cycle d'instruction.
- The following segment of the ASM flowchart for instruction cycle presents an initial configuration of the instruction cycle.
- D_7 = (l'opcode est 000 à 110) fait référence à une instruction de référence en mémoire.

- Si I = 1, alors l'instruction porte un indirect adresse et l'adresse effective de l'opérande est lue en la transférant à AR.

- Les quatre opérations ayant lieu à l'heure T_3 peuvent être symbolisées par:
  - $D'_7\, I\, T_3$: $AR \leftarrow M[AR]$
  - $D'_7\, I'\, T_3$: Nothing
  - $D_7\, I'\, T_3$: Execute a **register-reference** instruction
  - $D_7\, I\, T_3$: Execute an **input-output** instruction



Liste d'instructions d'ordinateurs de base (binaire)

D_7		I_{15}=0	I_{15}=1	op code			I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
0	*mem*	**MRI Direct Addr.**	MRI Indirect Addr.	*x*	*x*	*x*	Memory				A	D	D	R	E	S	S	
1	*reg*	**RRI**	**IOI**	**1**	**1**	**1**	C	o	d	e			Extension					

D_7	*Instr. Dec*	Addressing Mode I_{15}=0	I_{15}=1	op code I_{14}	I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
		MRI Direct Addr.	MRI Indirect Addr.				Memory				A	D	D	R	E	S	S	
D_7=0 ***mem***	D_0	**AND**=$0addr	AND_i =$8(addr)	*0*	*0*	*0*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_1	**ADD**=$1addr	ADD_i =$9(addr)	*0*	*0*	*1*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_2	**LDA** =$2addr	LDA_i =$A(addr)	*0*	*1*	*0*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_3	**STA** =$3addr	STA_i =$B(addr)	*0*	*1*	*1*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_4	**BUN** =$4addr	BUN_i =$C(addr)	*1*	*0*	*0*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_5	**BSA** =$5addr	BSA_i =$D(addr)	*1*	*0*	*1*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_6	**ISZ** =$6addr	ISZ_i =$E(addr)	*1*	*1*	*0*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
D_7=1 ***reg***		**RRI**	**IOI**	**1**	**1**	**1**	C	o	d	e			Extension					
	D_7	**CLA** =$7800	INP = $F800	*1*	*1*	*1*	**1**	0	0	0	0	0	0	0	0	0	0	0
	D_7	**CLE** =$7400	OUT = $F400	*1*	*1*	*1*	0	**1**	0	0	0	0	0	0	0	0	0	0
	D_7	**CMA** =$7200	SKI = $F200	*1*	*1*	*1*	0	0	**1**	0	0	0	0	0	0	0	0	0
	D_7	**CME** =$7100	SKO = $F100	*1*	*1*	*1*	0	0	0	1	0	0	0	0	0	0	0	0
	D_7	**CIR** =$7080	ION = $F080	*1*	*1*	*1*	0	0	0	0	**1**	0	0	0	0	0	0	0
	D_7	**CIL** =$7040	IOF = $F040	*1*	*1*	*1*	0	0	0	0	0	**1**	0	0	0	0	0	0
	D_7	**INC** =$7020	n/a	*1*	*1*	*1*	0	0	0	0	0	0	**1**	0	0	0	0	0
	D_7	**SPA** =$7010	n/a	*1*	*1*	*1*	0	0	0	0	0	0	0	**1**	0	0	0	0
	D_7	**SNA** =$7008	n/a	*1*	*1*	*1*	0	0	0	0	0	0	0	0	**1**	0	0	0
	D_7	**SZA** =$7004	n/a	*1*	*1*	*1*	0	0	0	0	0	0	0	0	0	**1**	0	0
	D_7	**SZE** =$7002	n/a	*1*	*1*	*1*	0	0	0	0	0	0	0	0	0	0	**1**	0
	D_7	**HLT** =$7001	n/a	*1*	*1*	*1*	0	0	0	0	0	0	0	0	0	0	0	**1**

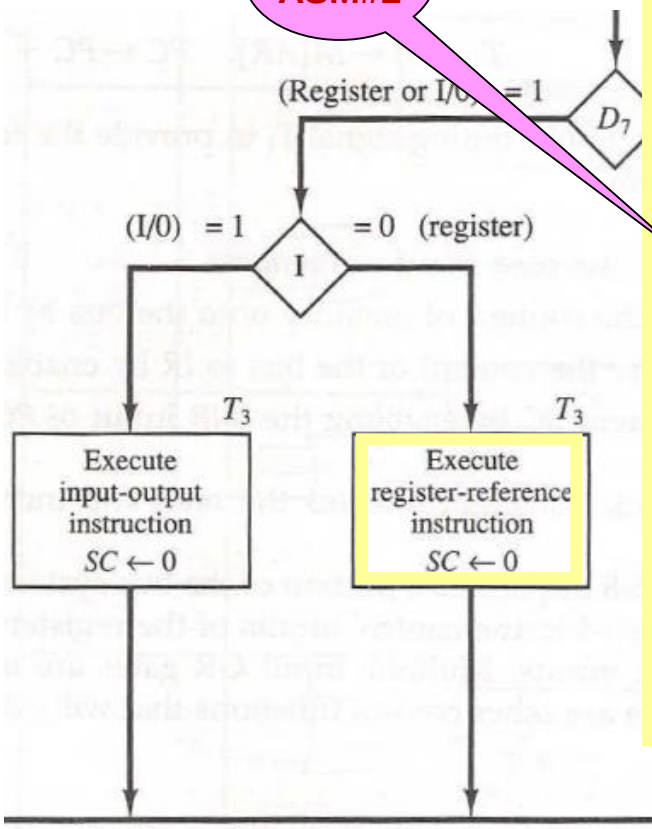
Binary encoding | **One-hot** (*bit-per-state*) encoding

"addr" = 12 bit address of the operand
"(addr)" = address of the operand address

Cycle d'instruction (EXEC)
Instructions de référence de registre (Register-Reference Instructions, RRI)

From the Instruction List: $I_{14}I_{13}I_{12} = 111 \Rightarrow D_7 = 1$
$I_{15} = I = 0$

RTL
ASM#2

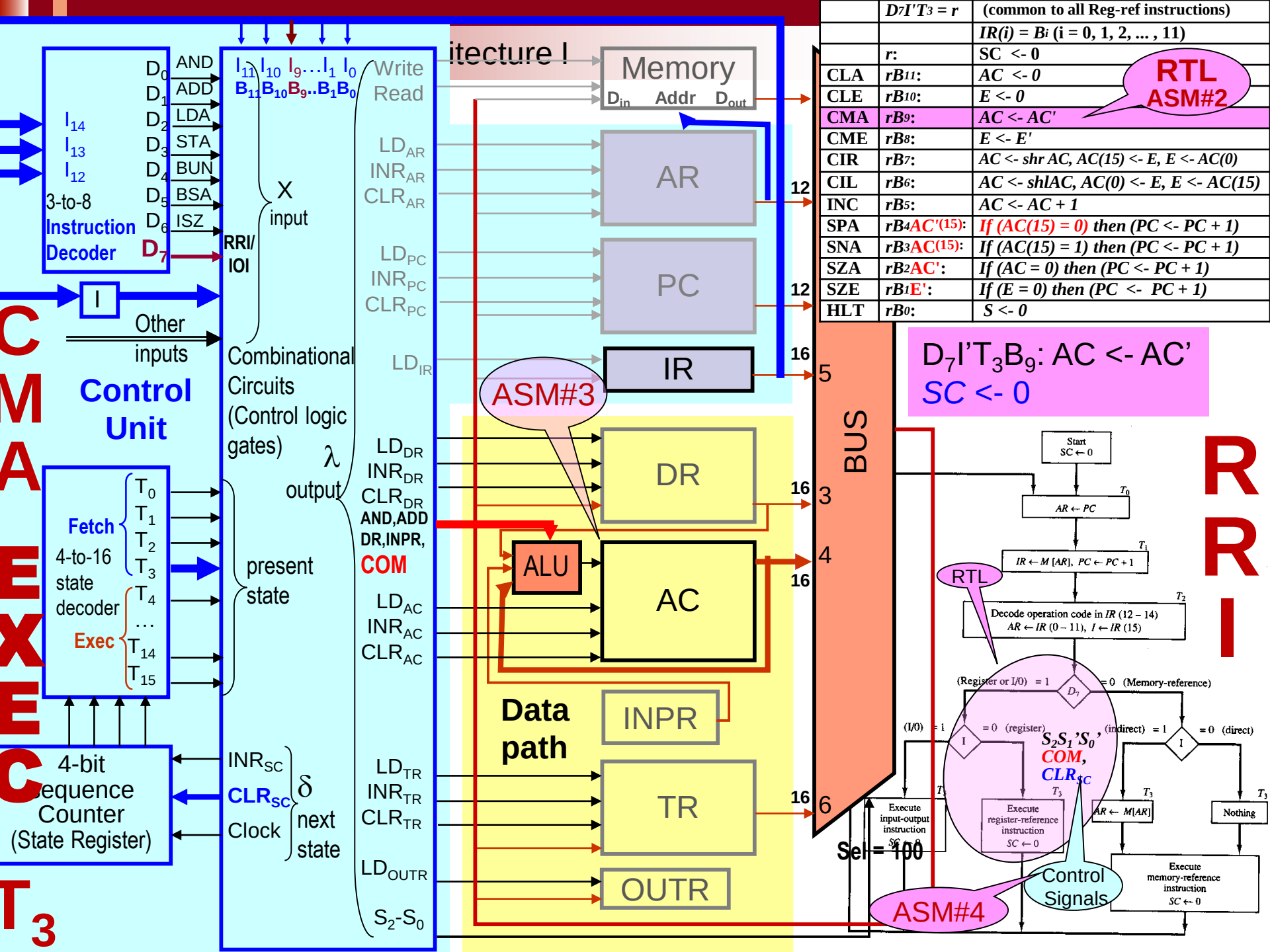


register

$D_7 I' T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0\text{–}11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

Every **R**egister-**R**eference **I**nstruction is executed during T_3



	$D_7I'T_3 = r$	(common to all Reg-ref instructions)
		$IR(i) = B_i$ (i = 0, 1, 2, ... , 11)
	r:	SC <- 0
CLA	rB_{11}:	AC <- 0
CLE	rB_{10}:	E <- 0
CMA	rB_9:	AC <- AC'
CME	rB_8:	E <- E'
CIR	rB_7:	AC <- shr AC, AC(15) <- E, E <- AC(0)
CIL	rB_6:	AC <- shlAC, AC(0) <- E, E <- AC(15)
INC	rB_5:	AC <- AC + 1
SPA	$rB_4AC'(15)$:	If (AC(15) = 0) then (PC <- PC + 1)
SNA	$rB_3AC(15)$:	If (AC(15) = 1) then (PC <- PC + 1)
SZA	rB_2AC':	If (AC = 0) then (PC <- PC + 1)
SZE	rB_1E':	If (E = 0) then (PC <- PC + 1)
HLT	rB_0:	S <- 0

Cycle d'instruction (EXEC)
Instructions de référence de mémoire (Memory-Reference Instructions, MRI)

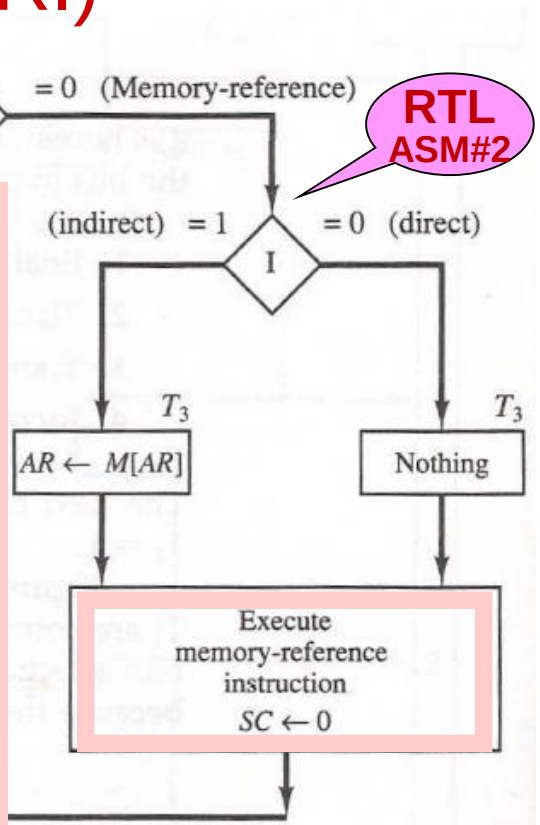
RTL ASM#2

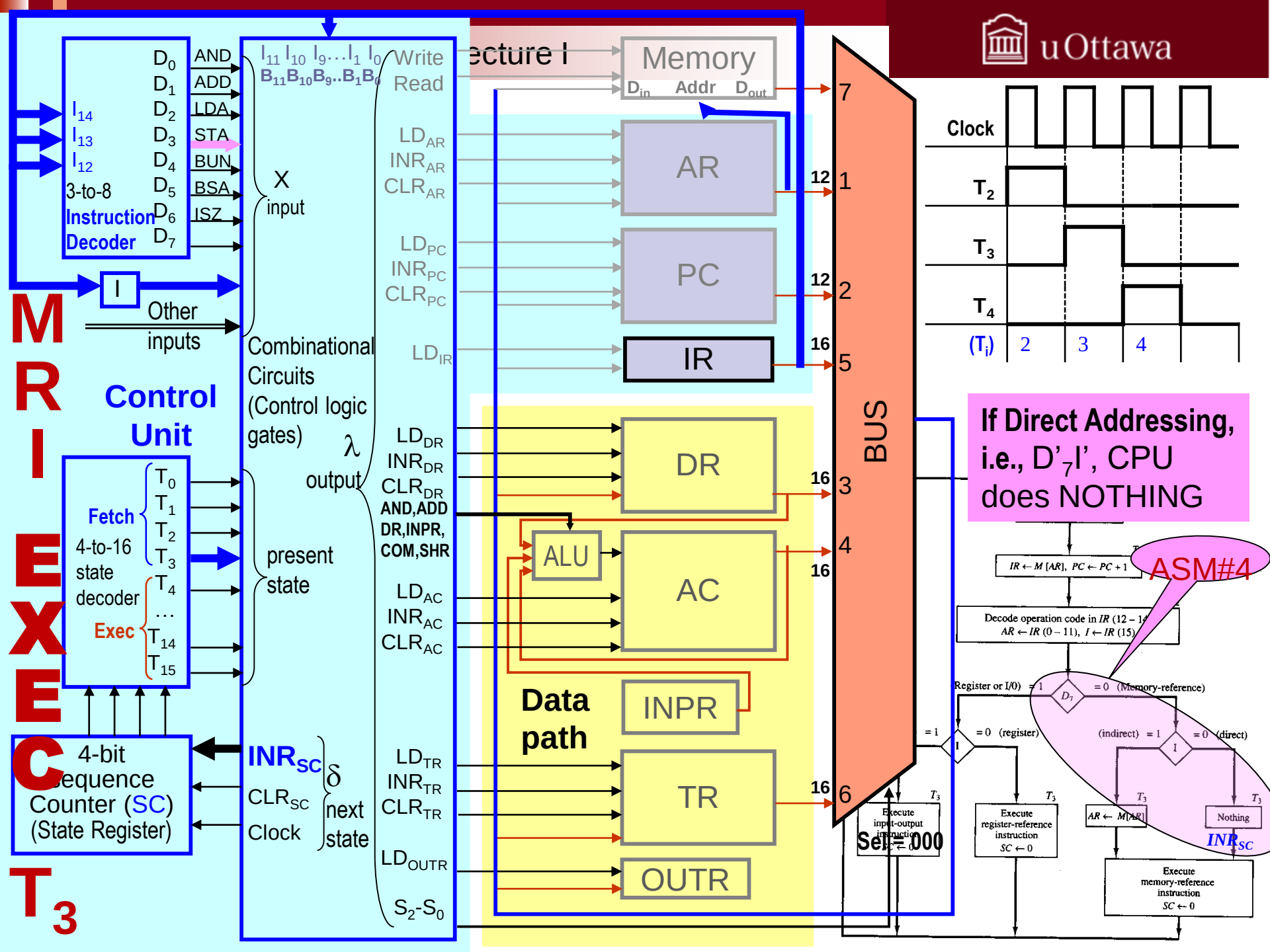
From the Instruction List:

$I_{14}I_{13}I_{12}$ = [000 – 110] => $D_i = 1$, i = {0 .. 6}

- $I_{15} = I = 0$ => direct addressing
- $I_{15} = I = 1$ => indirect addressing

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR]$, $E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$



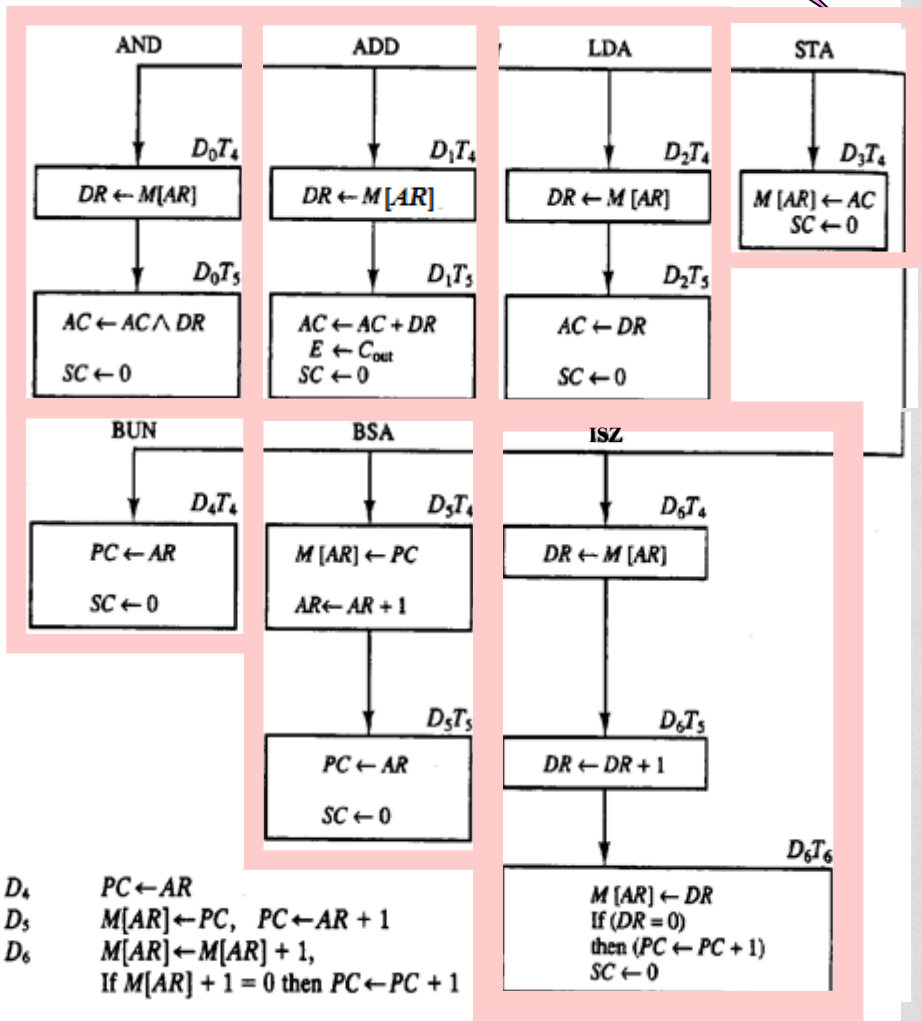

ecture I
Memory
Din Addr Dout
AR
PC
IR
DR
ALU
AC
INPR
TR
OUTR
BUS
Write
Read
LDAR INRAR CLRAR
LDPC INRPC CLRPC
LDIR
LDDR INRDR CLRDR
AND,ADD
DR,INPR,
COM,SHR
LDAC INRAC CLRAC
LDTR INRTR CLRTR
LDOUTR
S2-S0
I14 I13 I12
3-to-8
Instruction
Decoder
D0 AND
D1 ADD
D2 LDA
D3 STA
D4 BUN
D5 BSA
D6 ISZ
D7
I
Other
inputs
I11 I10 I9…I1 I0
B11B10B9..B1B0
X
input
Combinational
Circuits
(Control logic
gates)
λ
output
Control
Unit
Fetch
4-to-16
state
decoder
Exec
T0 T1 T2 T3
T4 … T14 T15
present
state
4-bit
Sequence
Counter (SC)
(State Register)
INRSC
CLRSC
Clock
δ
next
state
MRI
EXEC
T3
Clock
T2
T3
T4
(Ti) 2 3 4
If Direct Addressing,
i.e., D'7I', CPU
does NOTHING
ASM#4
IR ← M [AR], PC ← PC + 1
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
Register or I/O) = 1
D7
= 0 (Memory-reference)
= 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
Execute
input-output
instruction
SC ← 0
Execute
register-reference
instruction
SC ← 0
AR ← M[AR]
Nothing
INRSC
Execute
memory-reference
instruction
SC ← 0
Sel = 000

[AR]

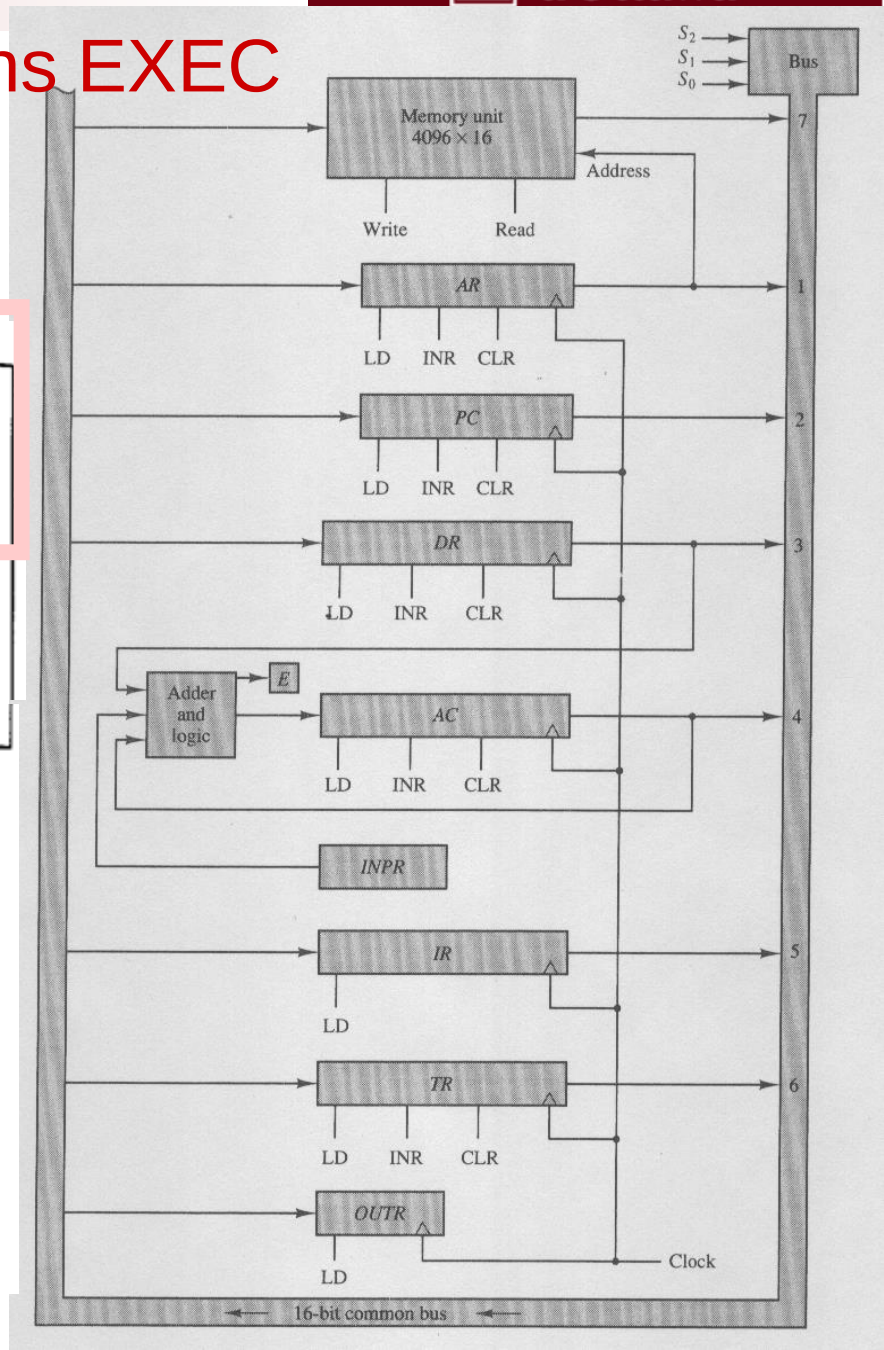
Memory-Reference Instructions EXEC

RTL ASM#2

AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR],\ E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$



BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC,\ PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

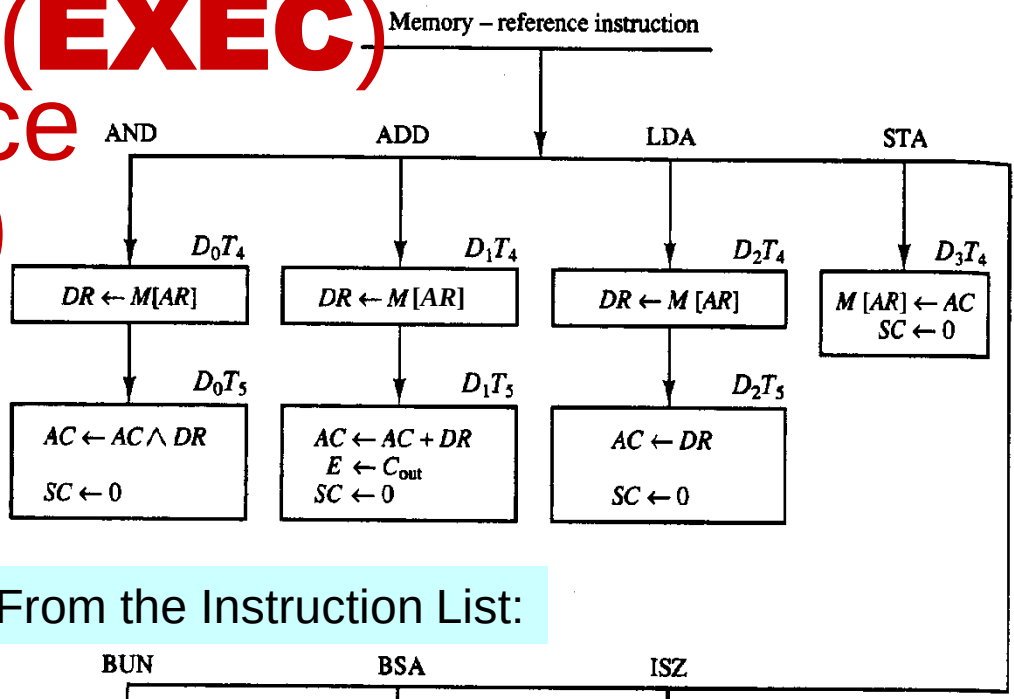


uOttawa

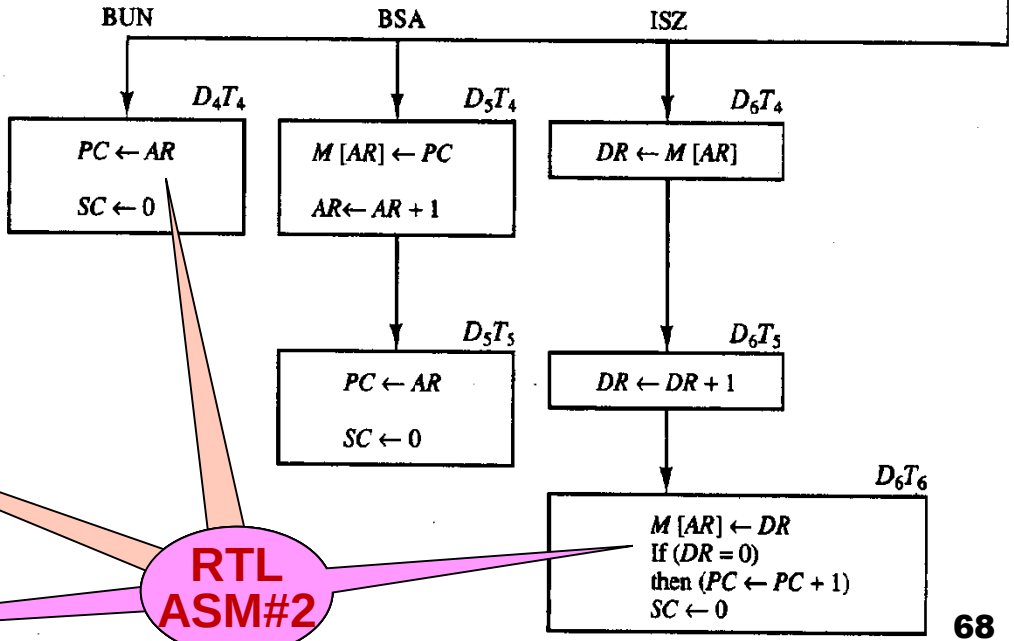
Instruction Cycle (EXEC) Memory-Reference Instructions (MRI)

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR]$, $E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

ASM#1



← From the Instruction List:



AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$,
	D_6T_6DR':	if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

RTL ASM#2

Note sur RTL

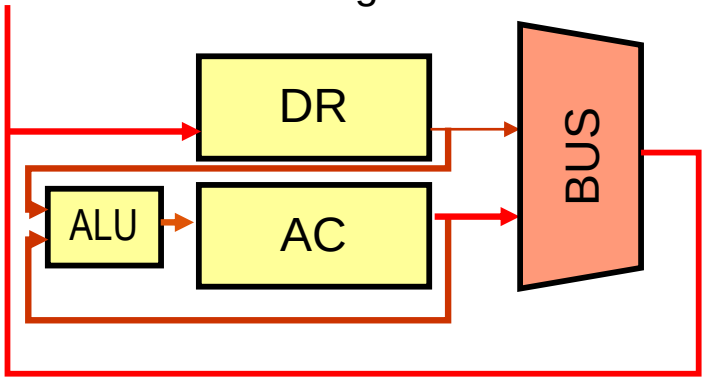
- Les instructions RTL doivent être lues comme ayant les valeurs présentes du côté droit de "=", tandis que le côté gauche fait référence au prochain état de ce registre; le transfert (charge de registre) a lieu sur le front montant de l'horloge.

- Exemple:

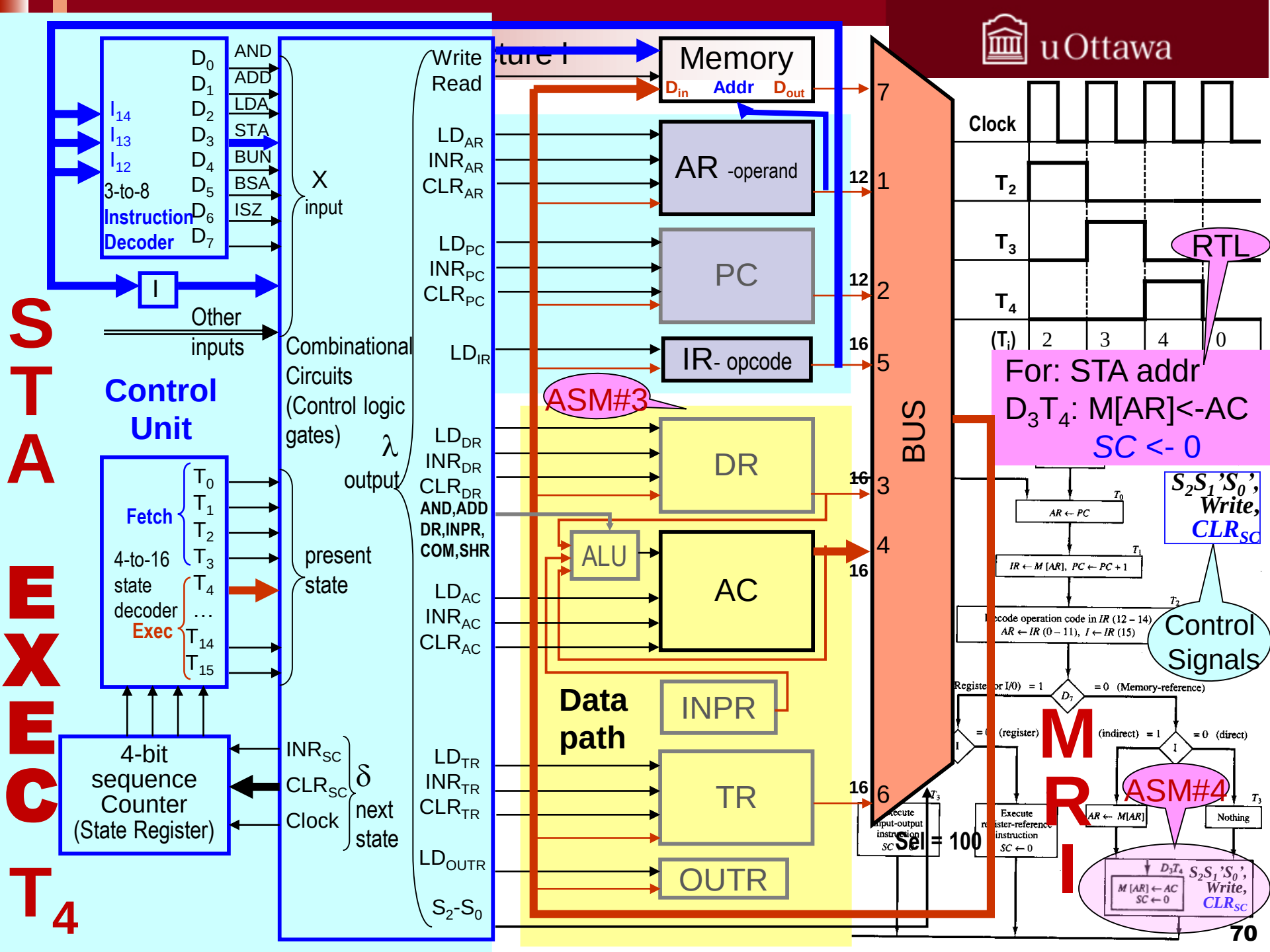
  D_1T_5: DR<- AC, AC<- AC + DR

  reads:

  D_1T_5: DR(n+1)<- AC(n)
  AC(n+1)<- AC(n) + DR(n)

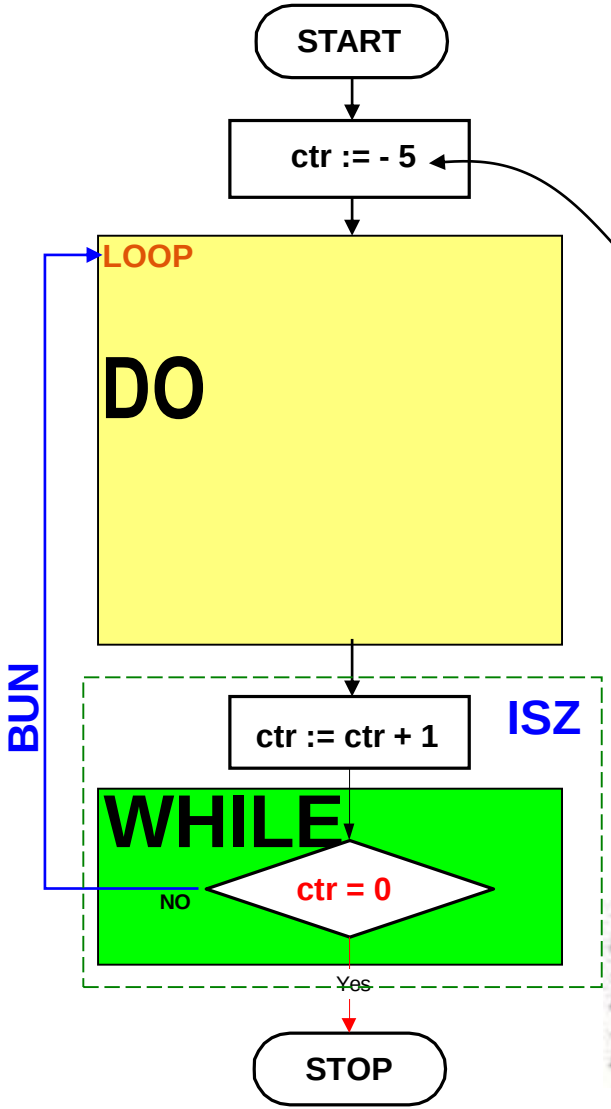


Même la deuxième RTL a du sens puisque **AC(n)+DR(n)** est la sortie de l'ALU qui doit être chargée dans AC sur le front montant de l'horloge. Après avoir chargé la nouvelle valeur dans AC, il y aura un délai jusqu'à ce que la sortie de l'ALU change, car ALU a besoin d'un peu de temps pour «déterminer» quel est le résultat le plus récent. En tant que tel, au moment où la sortie de l'ALU changera, le front montant aura disparu et rien ne sera autorisé dans AC jusqu'au prochain front montant!


STA EXEC T4
Control Unit
3-to-8 Instruction Decoder
I14
I13
I12
D0 AND
D1 ADD
D2 LDA
D3 STA
D4 BUN
D5 BSA
D6 ISZ
D7
I
Other inputs
X input
Combinational Circuits (Control logic gates)
λ output
Fetch
4-to-16 state decoder
Exec
T0
T1
T2
T3
T4
...
T14
T15
present state
4-bit sequence Counter (State Register)
INRSC
CLRSC
Clock
δ next state
Write
Read
LDAR INRAR CLRAR
LDPC INRPC CLRPC
LDIR
LDDR INRDR CLRDR
AND,ADD DR,INPR, COM,SHR
LDAC INRAC CLRAC
LDTR INRTR CLRTR
LDOUTR
S2-S0
Memory
Din Addr Dout
AR -operand
PC
IR- opcode
ASM#3
DR
ALU
AC
Data path
INPR
TR
OUTR
BUS
7
12 1
12 2
16 5
16 3
4
16
16 6
Sel = 100
Clock
T2
T3
T4
(Ti) 2 3 4 0
RTL
For: STA addr
D3T4: M[AR]<-AC
SC <- 0
S2S1'S0', Write, CLRSC
Control Signals
AR ← PC
IR ← M [AR], PC ← PC + 1
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
= 0 (Memory-reference)
(register)
(indirect) = 1
= 0 (direct)
MRI
ASM#4
AR ← M[AR]
Nothing
M [AR] ← AC
SC ← 0
D3T4 S2S1'S0', Write, CLRSC

ISZ - Incrémente et ignore si zéro
Mettre en œuvre DO-WHILE

DO
....
....
Increment ctr
ENDO
WHILE ctr < 0



MACHINE CODE		ASSEMBLY		
addr	memory			
000	...	LOOP	...	
001	...		...	
002	...		...	
003	6080		ISZ	CTR
004	2000		BUN	LOOP
005	7001		HLT	
...	...		...	
080	FFFB	CTR	equ -5	

if ctr = 0

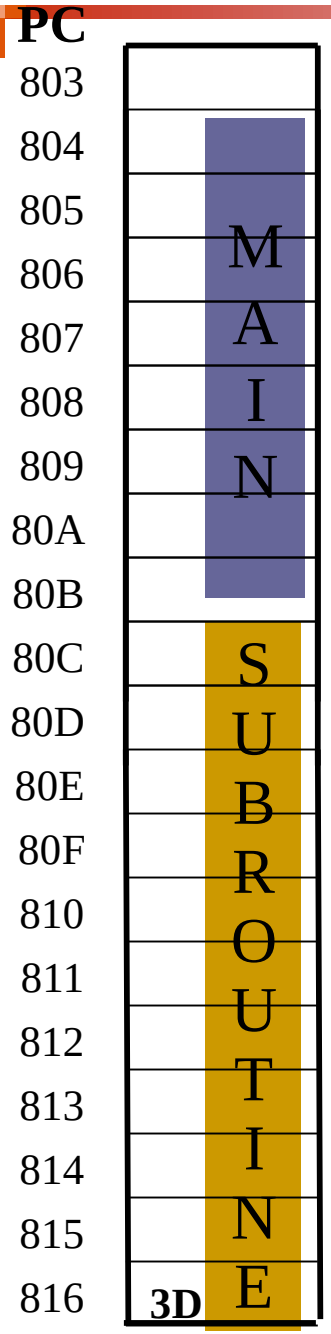
M
R
I

ASM#2

D_6T_4: $DR \leftarrow M[AR]$; $DR <= M[080] = FFFB$
D_6T_5: $DR \leftarrow DR + 1$; $DR <= FFFB + 1 = FFFC$
D_6T_6: $M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

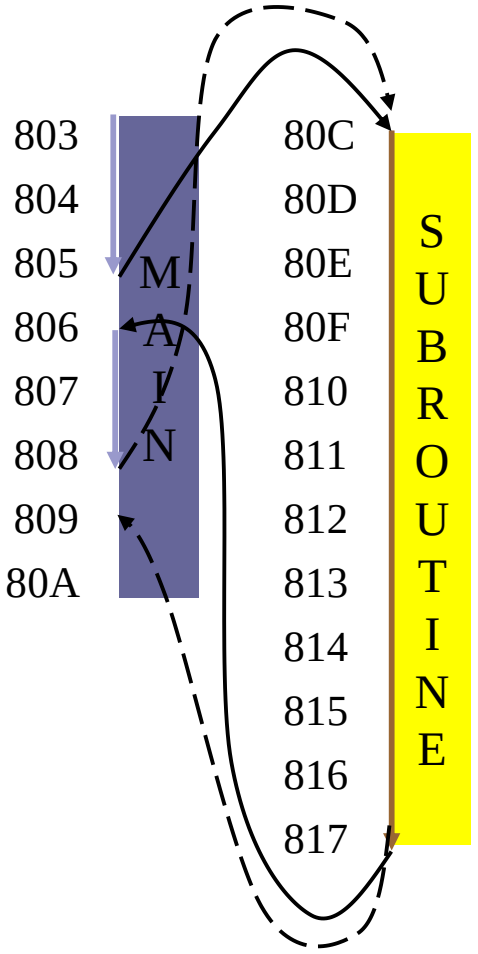
Comment …

1. … UNE SUBROUTINE FONCTIONNE-T-ELLE?
2. … DÉFINIR?
3. … UTILISER?



COMMENT ÇA MARCHE?

- Un sous-programme est un module de programme indépendant du programme principal.
- Pour l'utiliser, le programme principal transfère le contrôle au sous-programme.
- Le sous-programme remplit sa fonction puis renvoie le contrôle au programme principal.
- Problème principal: définir un mécanisme pour stocker le



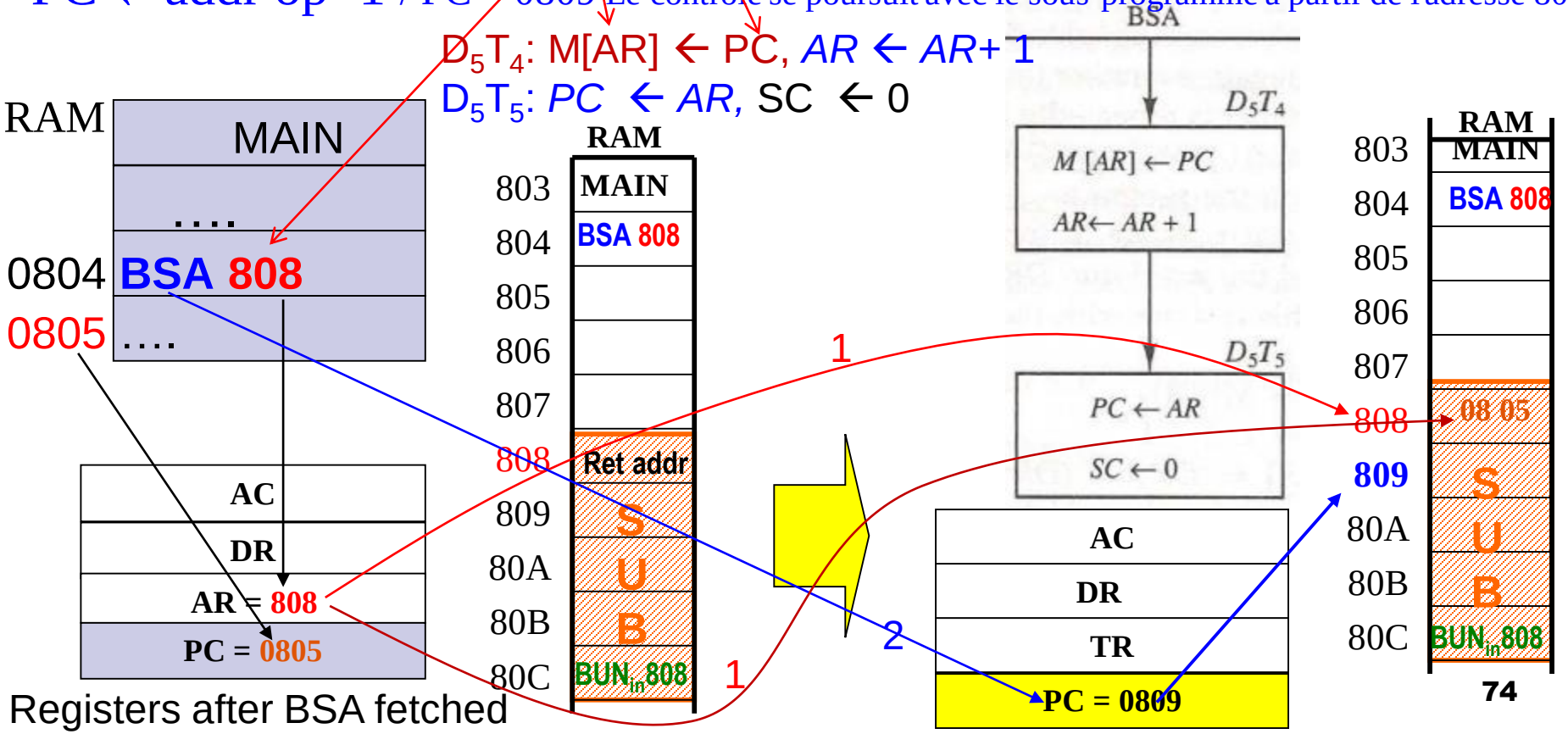
ADRESSE DE RETOUR, de sorte que le sous-programme sache où le trouver lorsqu'il revient au programme appelant

MAIN **APPELLE** LA SOUS-ROUTINE (**BSA**)

L'ADRESSE RETOUR est stockée à l'emplacement de mémoire indiqué par BSA. Le sous-programme commence à l'adresse suivante.

1. m(addr op)←(PC) /m(808) = 0805 = l'adresse de retour 805 est stockée à l'emplacement 808
PC ←addr op+1 / PC = 0809 Le contrôle se poursuit avec le sous-programme à partir de l'adresse 809

D_5T_4: M[AR] ← PC, *AR* ← *AR*+ 1
D_5T_5: *PC* ← *AR*, SC ← 0



Registers after BSA fetched

SOUS-ROUTINE **RETOURNE** À MAIN

RAM

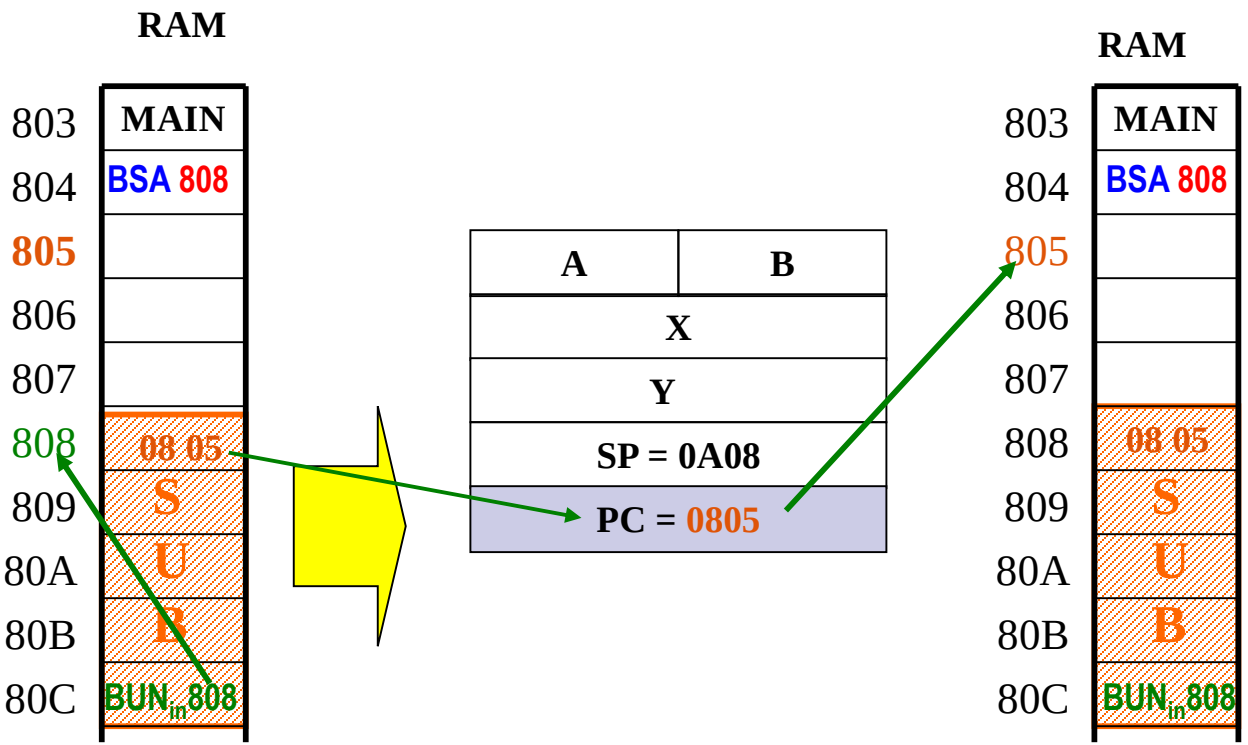
	Subroutine
080B	…
080C	$BUN_{indirect}$ 808
080D	….

AC
DR
TR
PC = 080D

1

3. PC ← [m(m(808))] PC=0805

$BUN_{indirect}$ 808: $D_7'T_3$: AR ← M[AR]
D_4T_4: *PC* ← *AR*, SC ← 0



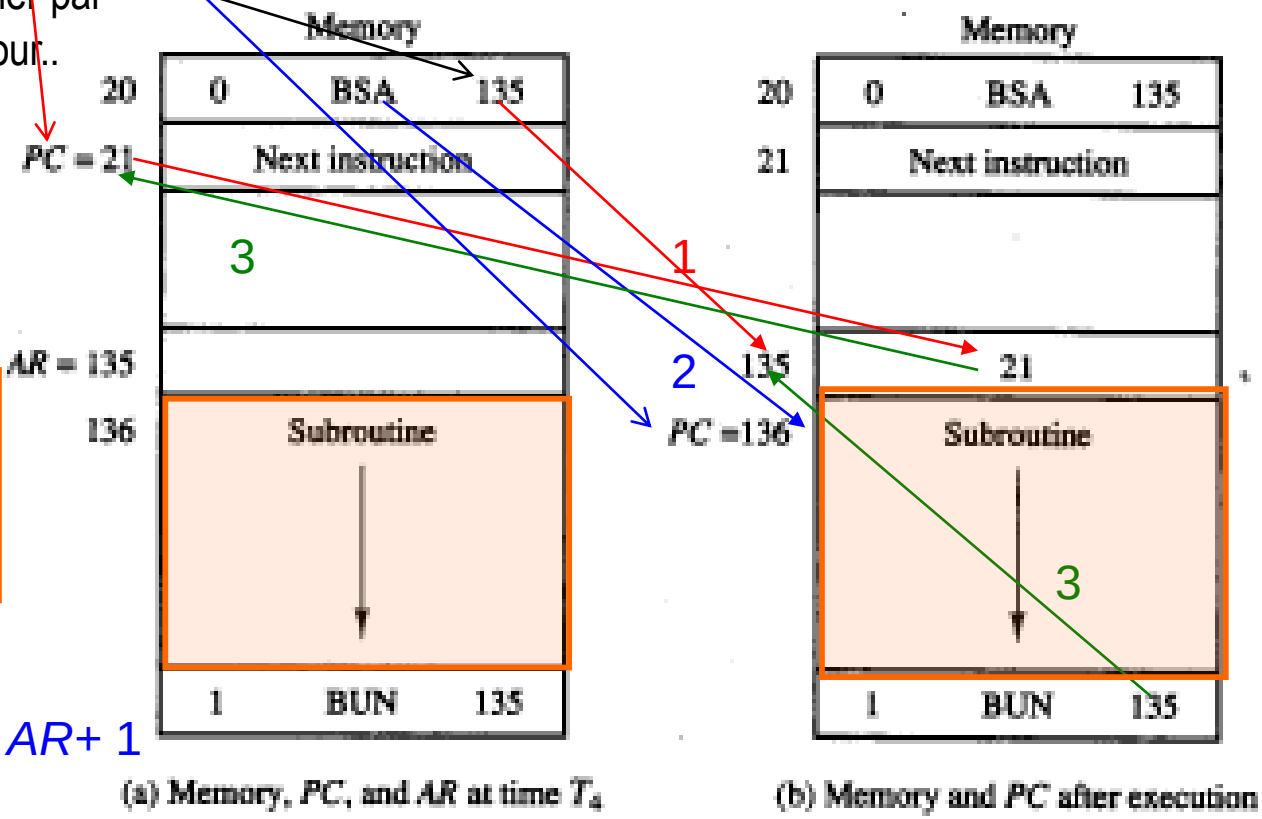
BSA: Branch and Save Return Address

L'instruction BSA

1. *stocke l'**adresse de retour** dans l'emplacement de mémoire spécifié par son adresse effective (135). L'**adresse de retour** = l'adresse de l'instruction à exécuter après l'exécution du sous-programme = 21 dans notre exemple; cette adresse est déjà préparée sur PC*;
2. branche vers la première instruction du sous-programme qui est stockée dans la mémoire à l'adresse suivante (136) après l'adresse de retour stockée..
3. Le sous-programme doit se terminer par un BUN indirect à l'adresse de retour..

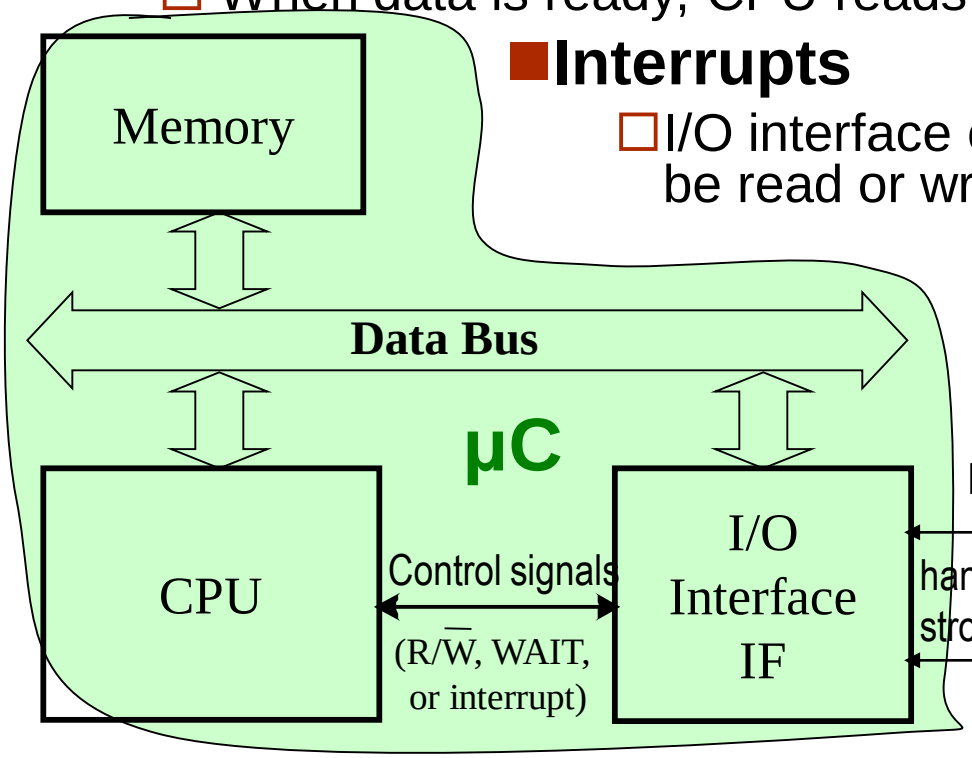
20	BSA 135
21	next instruction
22	……
….	…….
135	stored ret. addr (21)
136	Subroutine 1st instruc
137	…..
….	…..
159	Subroutine last instr
160	$BUN_{indirect}$ 135

1) D_5T_4: M[AR] ← PC, *AR* ← *AR*+ 1
2) D_5T_5: *PC* ← *AR*, SC ← 0

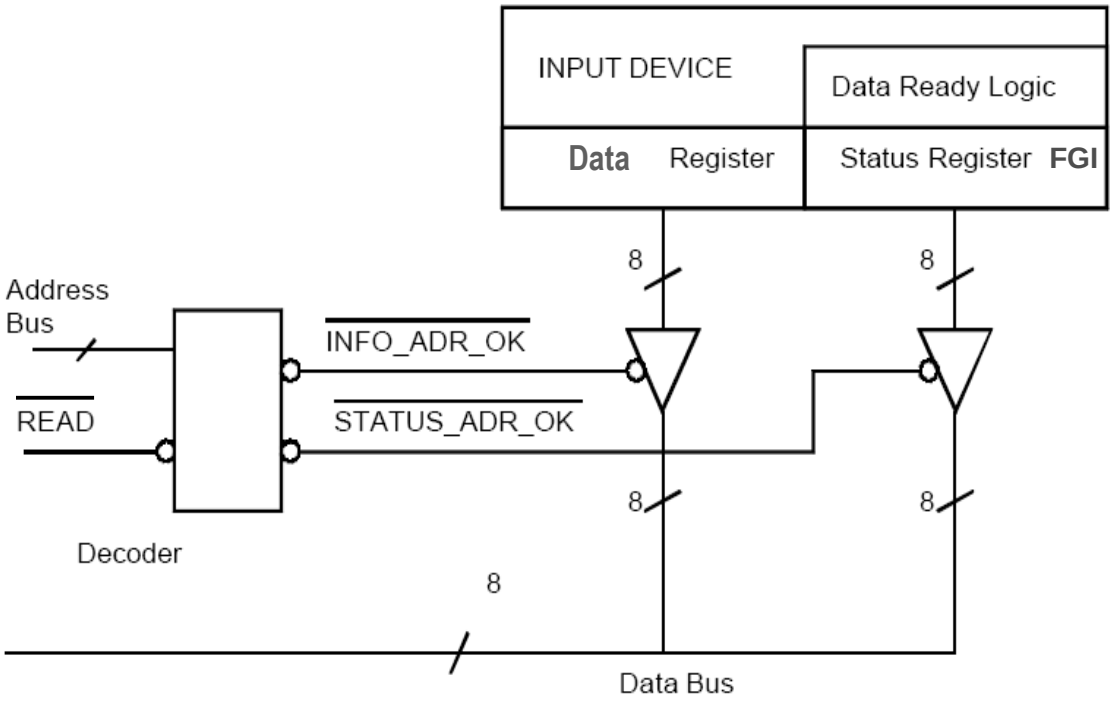


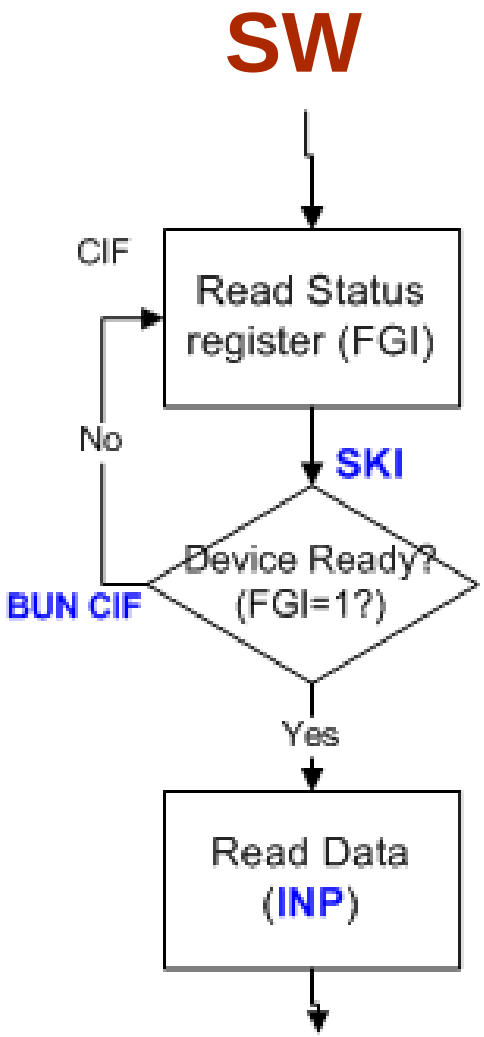
CPU & I/O Interface

- Hardware provides an **interface** to the I/O device. Contains:
  - **data register (**for data transfer, here: INPR and OUTR)
  - **status register** (flags=peripheral ready, here: FGI & FGO)
  - **control register** (here: R, IEN)
- **SW Polling** (Programmed Control)
  - CPU monitors the status register
  - When data is ready, CPU reads or writes from/to the data register
- **Interrupts**
  - I/O interface can interrupt the CPU when data can be read or written (will study later)
- **DMA (**direct-memory access**):**
  - the IF takes over the control of the system's Data, Address & Control Buses and transfers sequences of data direct to the memory.



Software Polling





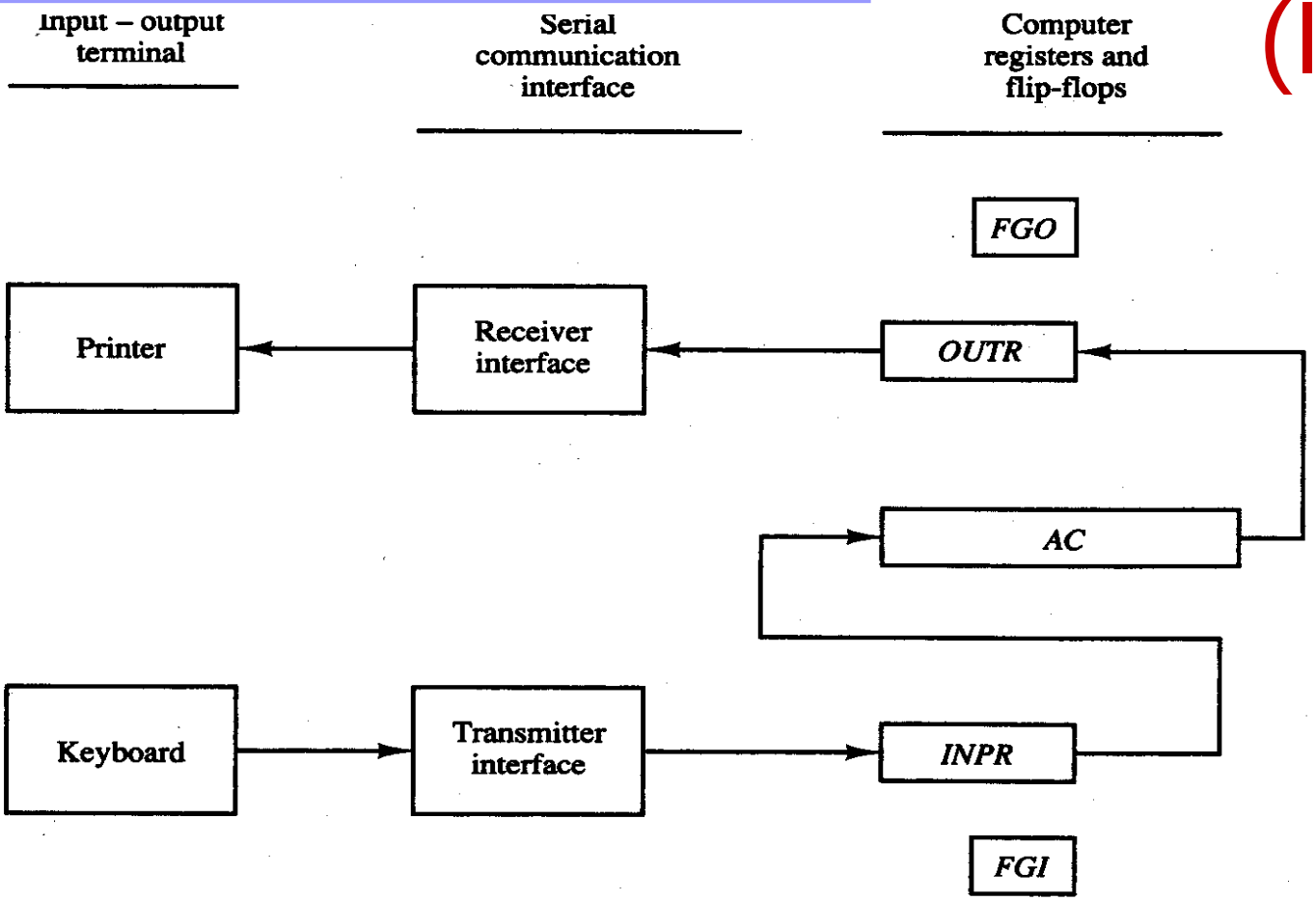
Basic Computer Input/Output Instructions (IOI)

$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6–11)$ that specifies the instruction]

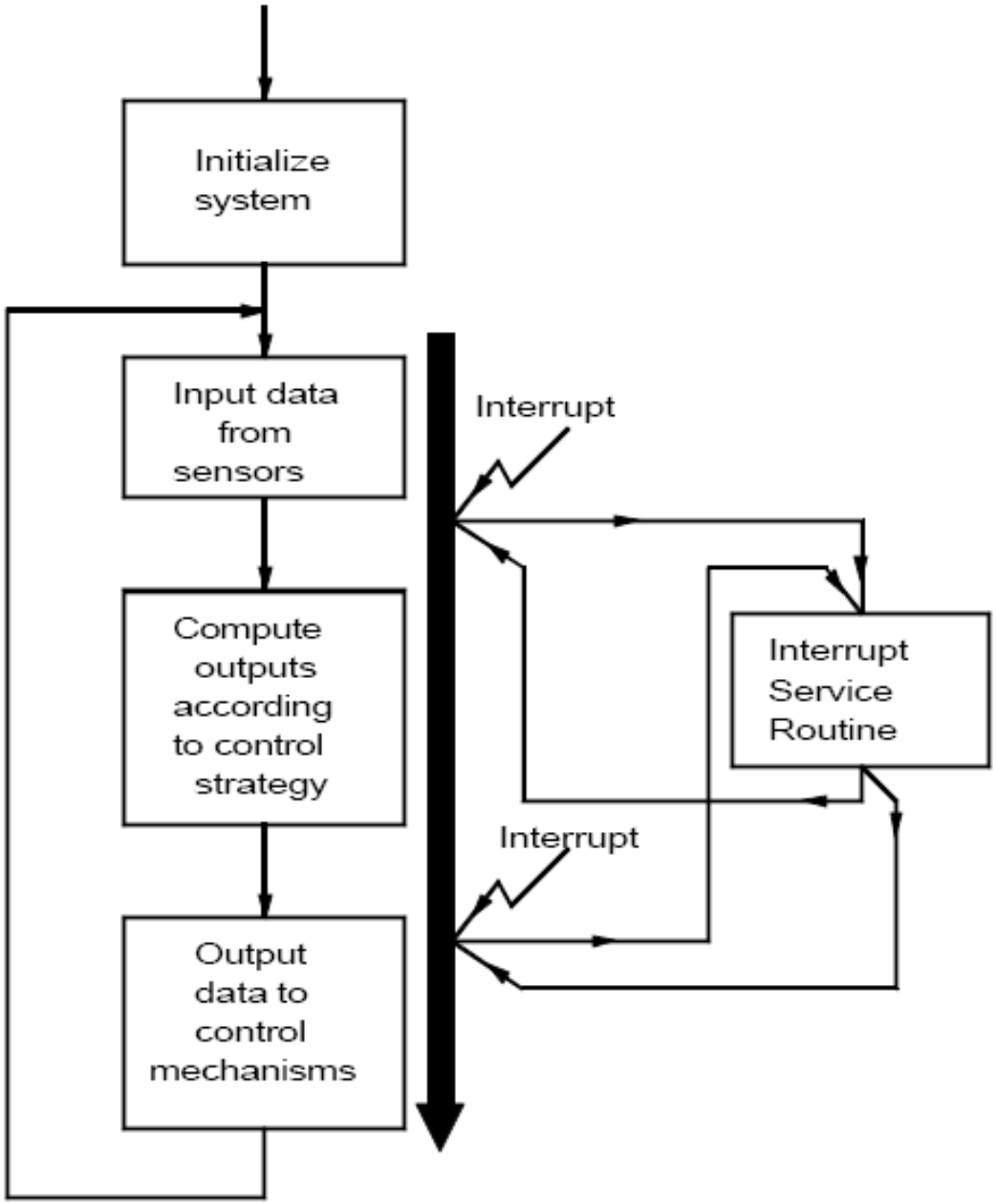
peripheral

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$	Output character
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

RTL

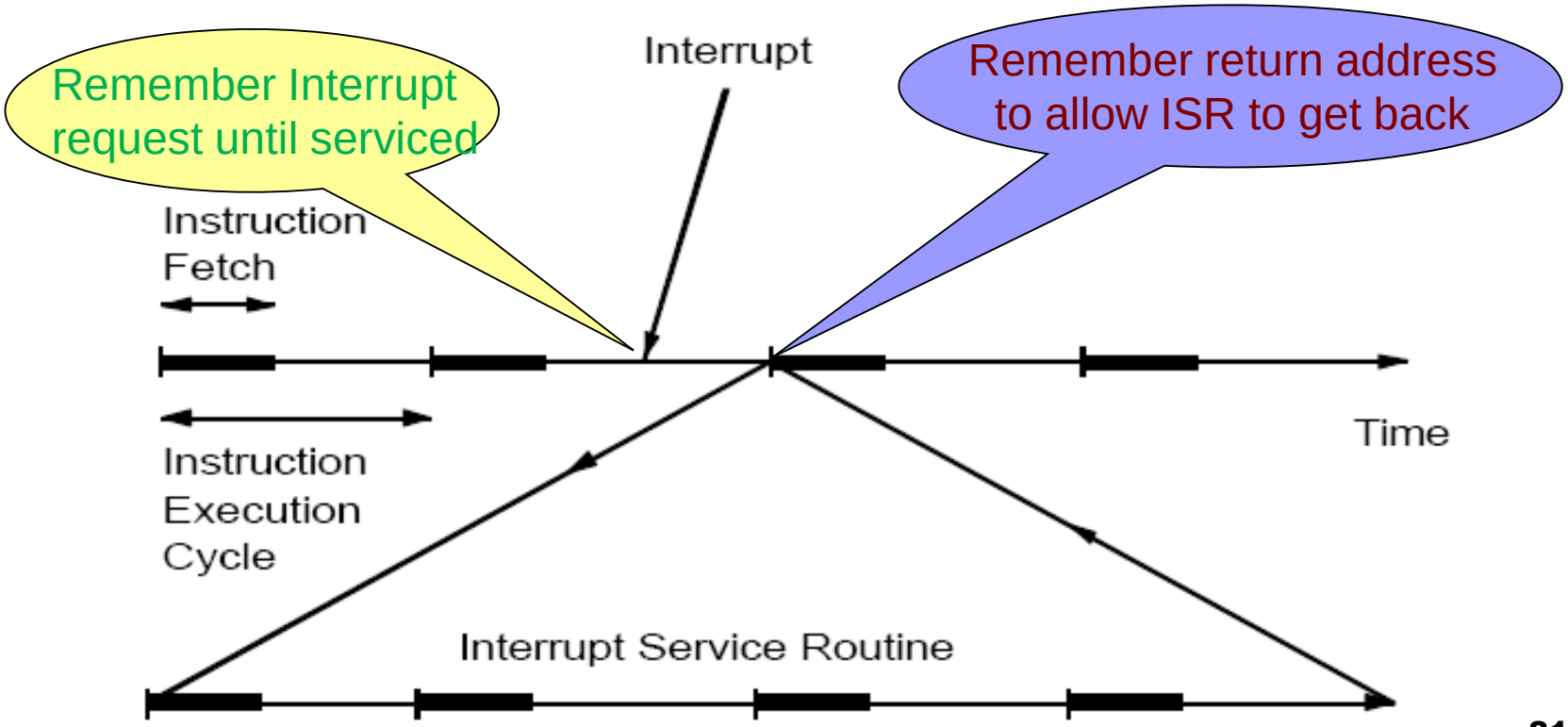


Concepts fondamentaux de l'interruption



Synchronisation interne du processeur asynchrone

- Les interruptions peuvent survenir à tout moment pendant un cycle d'instruction

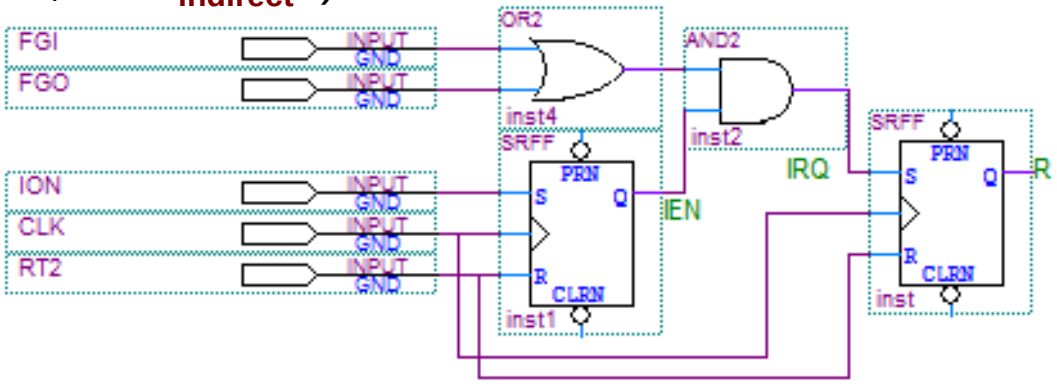


Spécifications du système d'interruption

- Permet aux événements asynchrones de se produire et d'être reconnus..
- Attendez que l'instruction en cours se termine avant de traiter une interruption.
- Service d'interruption avec une sous-routine(*Interrupt Service Routine* = *ISR*) et retour au code interrompu.
- Activation et désactivation des interruptions (IEN) (IEN)
- Sources multiples d'interruptions (ici 2: FGO, FGI)
  - Interruptions simultanées..

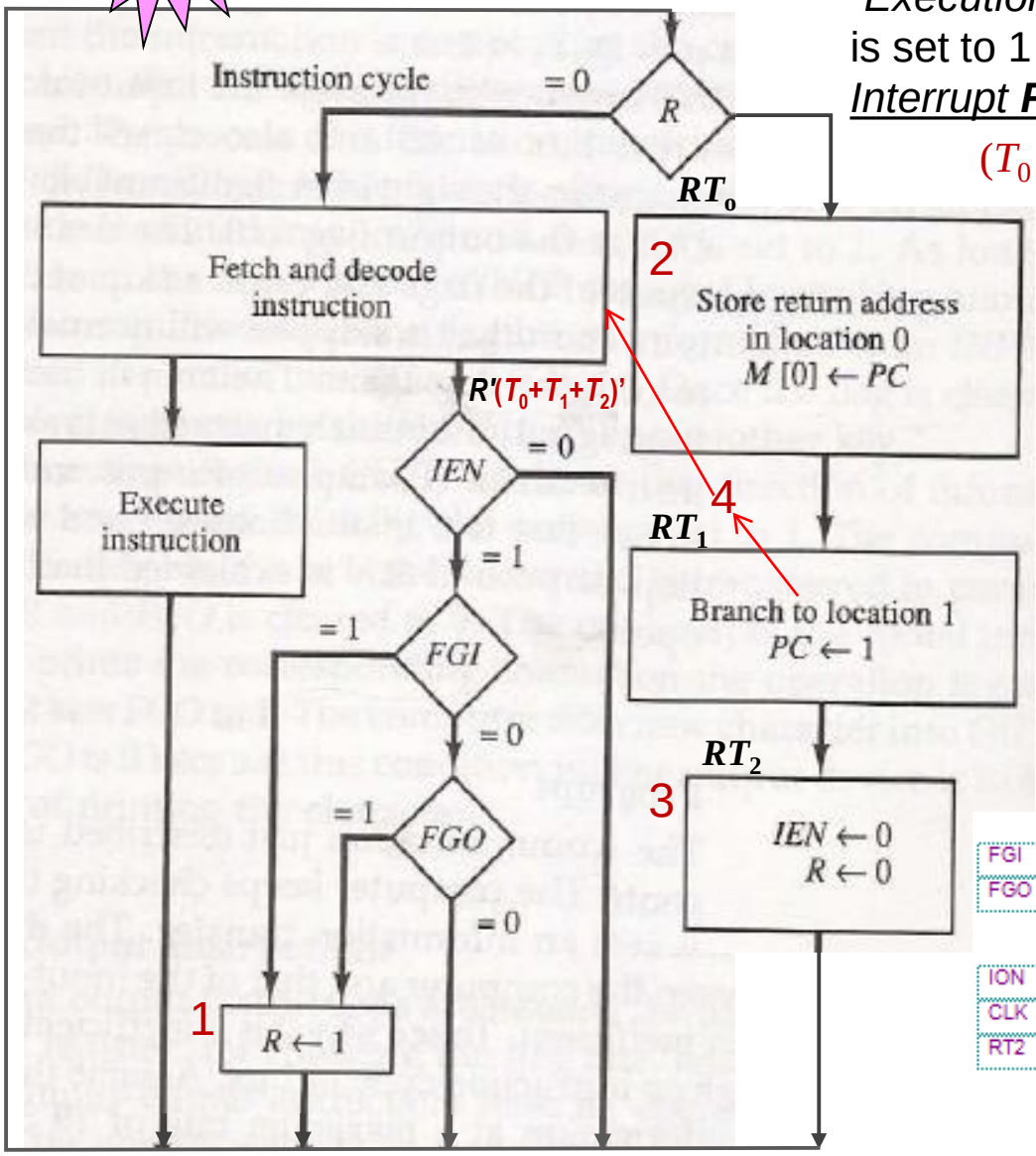
Interruption matérielle interne du processeur

- Use a flip-flop (R) to catch IRQ (FGI, FGO)
  - Multiple device interrupt lines (FGI, FGO) can be OR-wired together
  - CPU waits until the current instruction is executed and then can process interrupt
- Can enable/disable interrupt using enable-disable flip-flop (IEN)
  - When interrupt is acknowledged by CPU HW, interrupts are disabled (IEN ← 0) for the duration of interrupt servicing
- The source of interrupt is determined by FGI or FGO → the Interrupt Service Routine (ISR) has to check which I/O device generated the IRQ
- The return address is stored at address 0; at the end of ISR
  - ION is executed to arm the interrupt system IEN ← 1
  - **BUN 0 I** (i.e., **BUN$_{indirect}$0**) is executed to return to the interrupted program



Cycle d'interruption

RTL

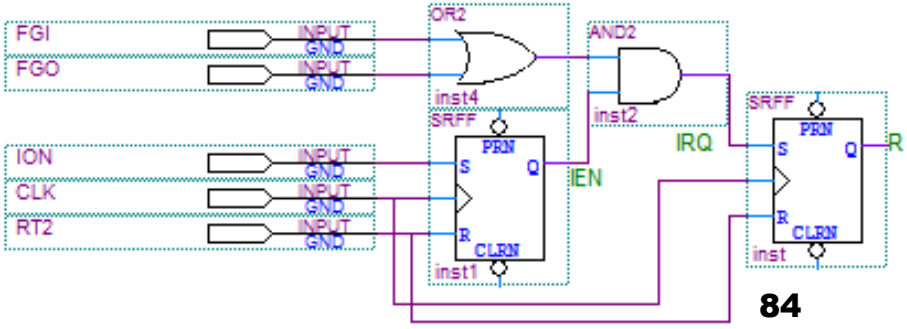


Once the "*Fetch*" phase is finished, during the "*Execution*" phase $(T_0+T_1+T_2)'$, if a flag (***FGI*** or ***FGO***) is set to 1 by an interrupt, the computer stores the *Interrupt* ***R****equest* in ***R*** if enabled by ***IEN*** :

$(T_0 + T_1 + T_2)'(IEN)\ (FGI + FGO)$: $R \leftarrow 1$

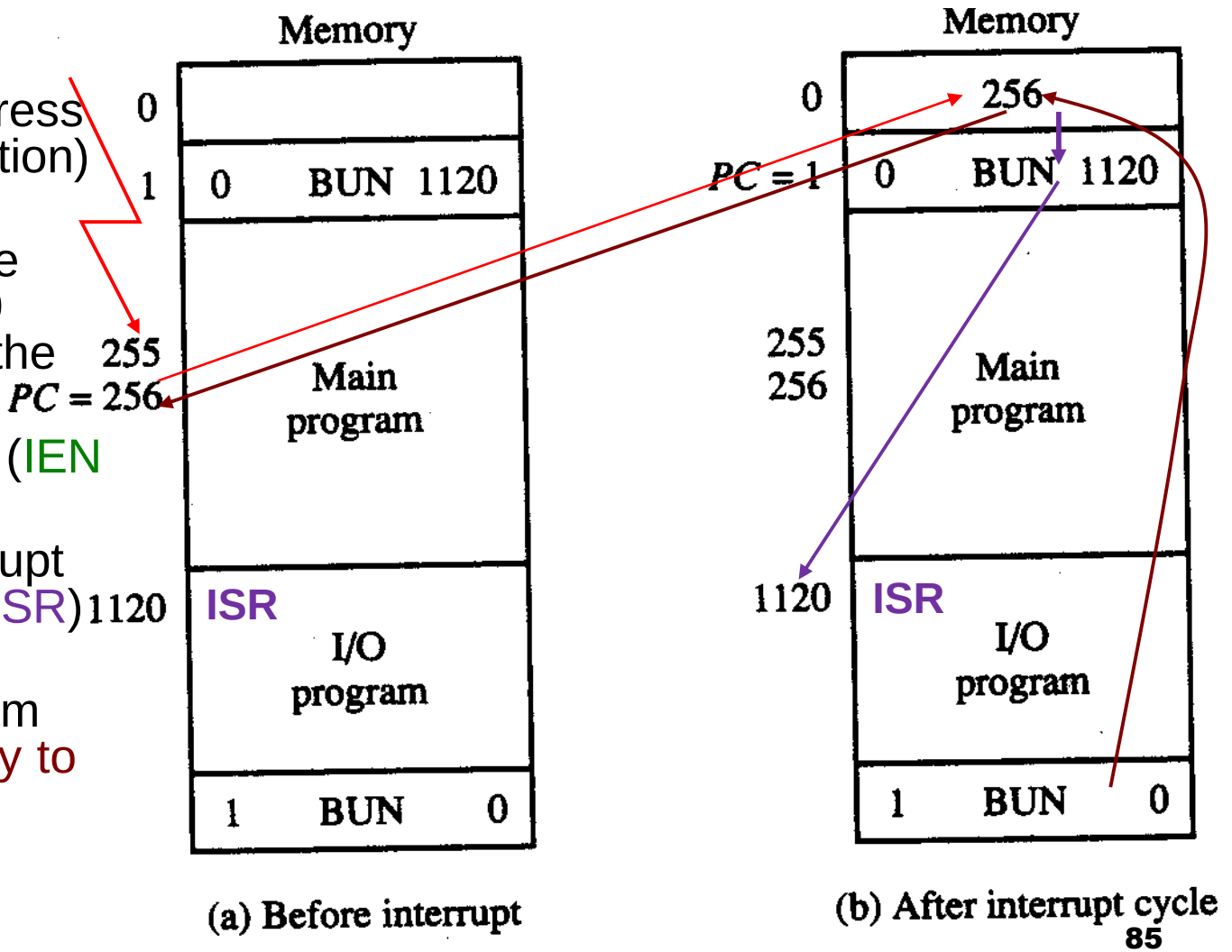
and completes the execution of the instruction in progress;

2. Next goes to an Interrupt cycle, where stores the return address in the location 0 of the memory;
3. Then disables the interrupt system (***IEN***=0) and resets *R* (since the system already takes care of this interrupt);
4. Executes the instruction stored in the memory location 1, which is a direct unconditional branching to the base address of the service routine.

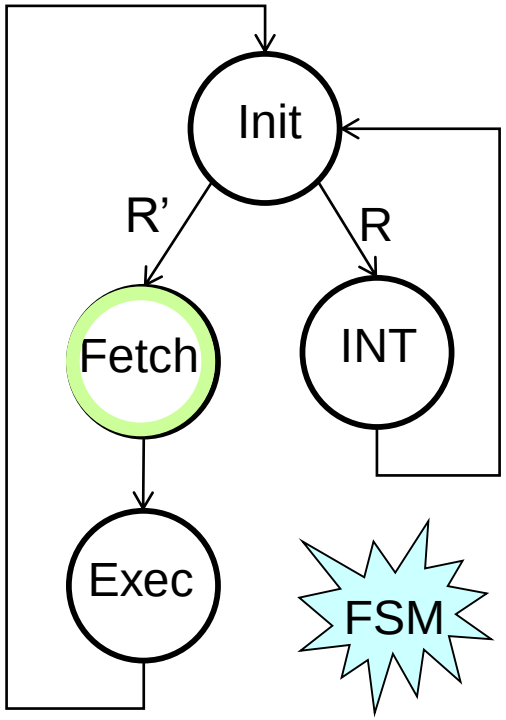


Interrupt Service Cycle (INT)

1. Save the return address (PC=address of the next instruction) at the address 0;
2. BUN directly to the **ISR** address 1120 corresponding to the interrupt
3. Disable interrupts (IEN ← 0, R ← 0)
4. Execute the Interrupt Service Routine (ISR)
5. Resume the interrupted program with BUN indirectly to address 0



Fonctionnement de l'ordinateur de base



RTL

Start
$SC \leftarrow 0, IEN \leftarrow 0, R \leftarrow 0$

R: (instruction cycle) = 0 / = 1 (interrupt cycle)

$R'T_0$: $AR \leftarrow PC$

$R'T_1$: $IR \leftarrow M[AR], PC \leftarrow PC + 1$

$R'T_2$: $AR \leftarrow IR(0-11), I \leftarrow IR(15)$
$D_0 \cdots D_7 \leftarrow$ Decode $IR(12-14)$

D_7: (Register or I/0) = 1 / = 0 (Memory – reference)

I: (I/0) = 1 / = 0 (register)

$D_7 I T_3$: Execute input-output instruction (Table 5-5)

$D_7 I' T_3$: Execute register-reference instruction (Table 5-3)

I: (indirect) = 1 / = 0 (direct)

$D'_7 I T_3$: $AR \leftarrow M[AR]$

$D'_7 I' T_3$: Nothing

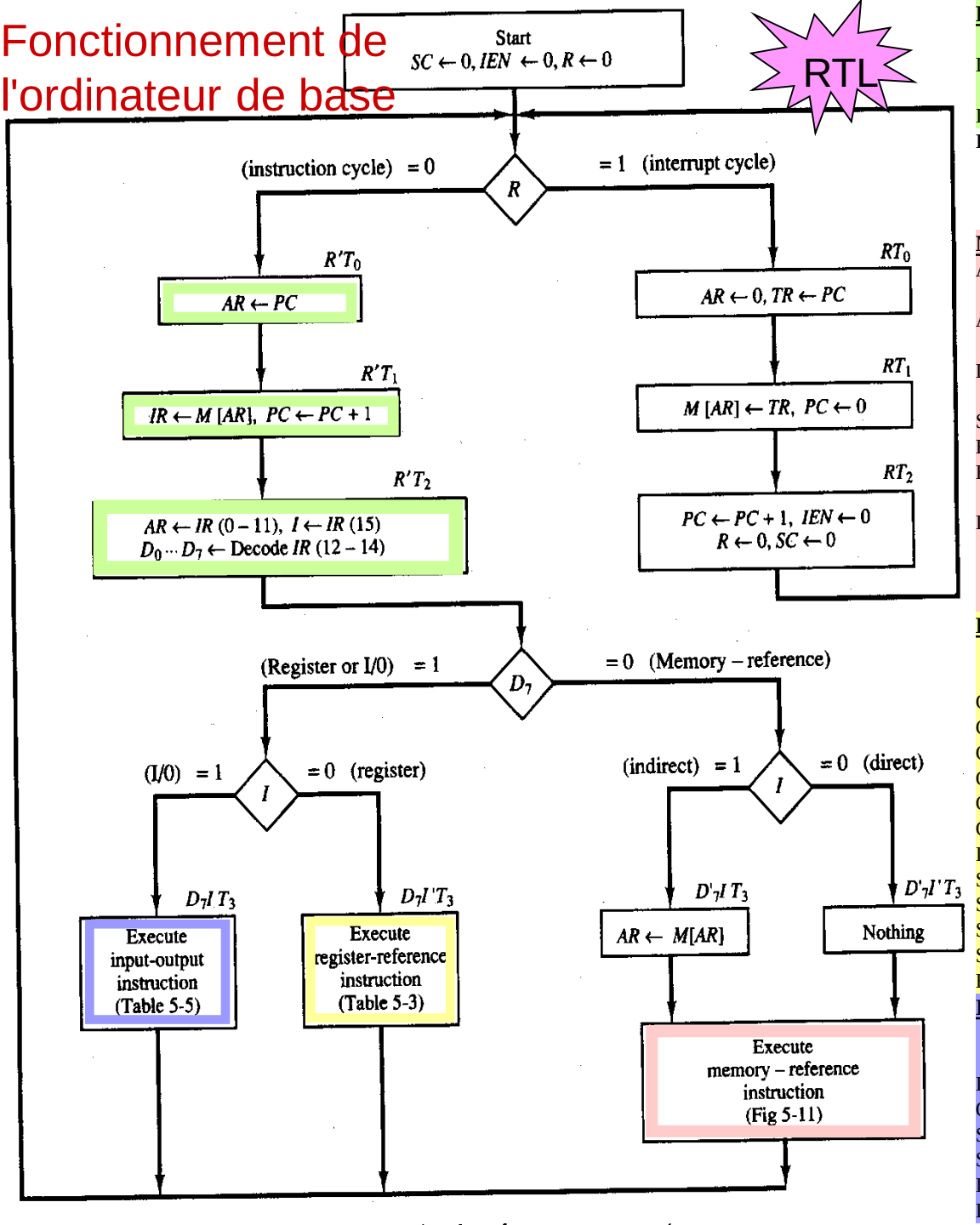
Execute memory – reference instruction (Fig 5-11)

RT_0: $AR \leftarrow 0, TR \leftarrow PC$

RT_1: $M[AR] \leftarrow TR, PC \leftarrow 0$

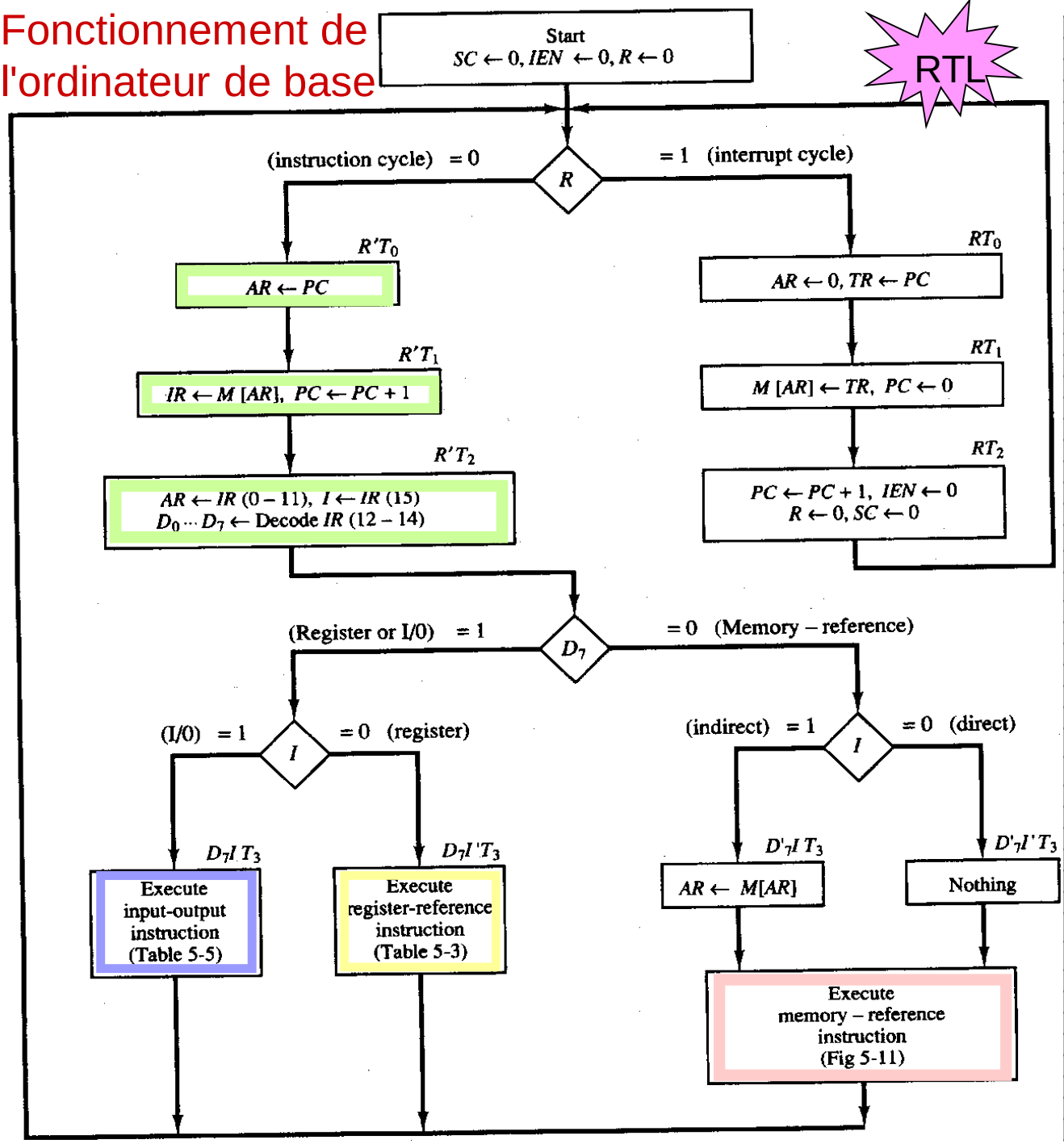
RT_2: $PC \leftarrow PC + 1, IEN \leftarrow 0$
$R \leftarrow 0, SC \leftarrow 0$

Fonctionnement de l'ordinateur de base

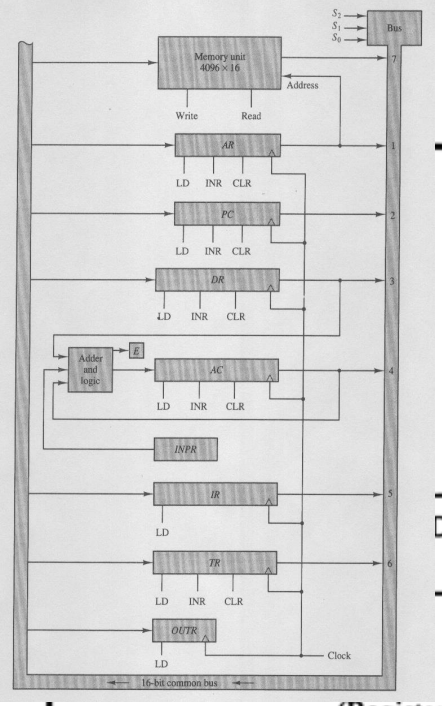


Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$, $AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Interrupt:	$T_0\overline{T_1}\overline{T_2}(IEN)(FGI + FGO)$:	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR,$
	D_6T_6DR':	if $(DR = 0)$ then $(PC \leftarrow PC + 1), SC \leftarrow 0$
Reg-ref	$D_7I'T_3 = r$	(common to all reg.-ref. instr.)
		$IR(i) = B_i$ (i = 0, 1, 2, ... , 11)
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow$ shr $AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow$ shl$AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	$rB_4AC'(15)$:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	$rB_3AC(15)$:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2AC':	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$
SZE	rB_1E':	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
In-out	$D_7IT_3 = p$	(common to all input-output instructions)
	$IR(i) = B_i$	(i = 6, 7, 8, 9, 10, 11)
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$
SKI	pB_9FGI:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8FGO:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

Fonctionnement de l'ordinateur de base



Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$,
		$AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Interrupt	$\overline{T_0}\overline{T_1}\overline{T_2}$ IEN (FGI+FGO):	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
Execute Memory-reference Instructions (MRI)		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR,$
	D_6T_6DR':	if $(DR = 0)$ then $(PC \leftarrow PC + 1), SC \leftarrow 0$
Execute Register-reference Instructions (RRI)		
	$D_7I'T_3 = r$	(common to all Register-ref. instructions)
		$IR(i) = B_i$ (i = 0, 1, 2, ... , 11)
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl\ AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	$rB_4AC'(15)$:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	$rB_3AC(15)$:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2AC':	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$
SZE	rB_1E':	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
Execute Input-output (IOI)		
	$D_7IT_3 = p$	(common to all input-output instructions)
	$IR(i) = B_i$	(i = 6, 7, 8, 9, 10, 11)
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$
SKI	pB_9FGI:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8FGO:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

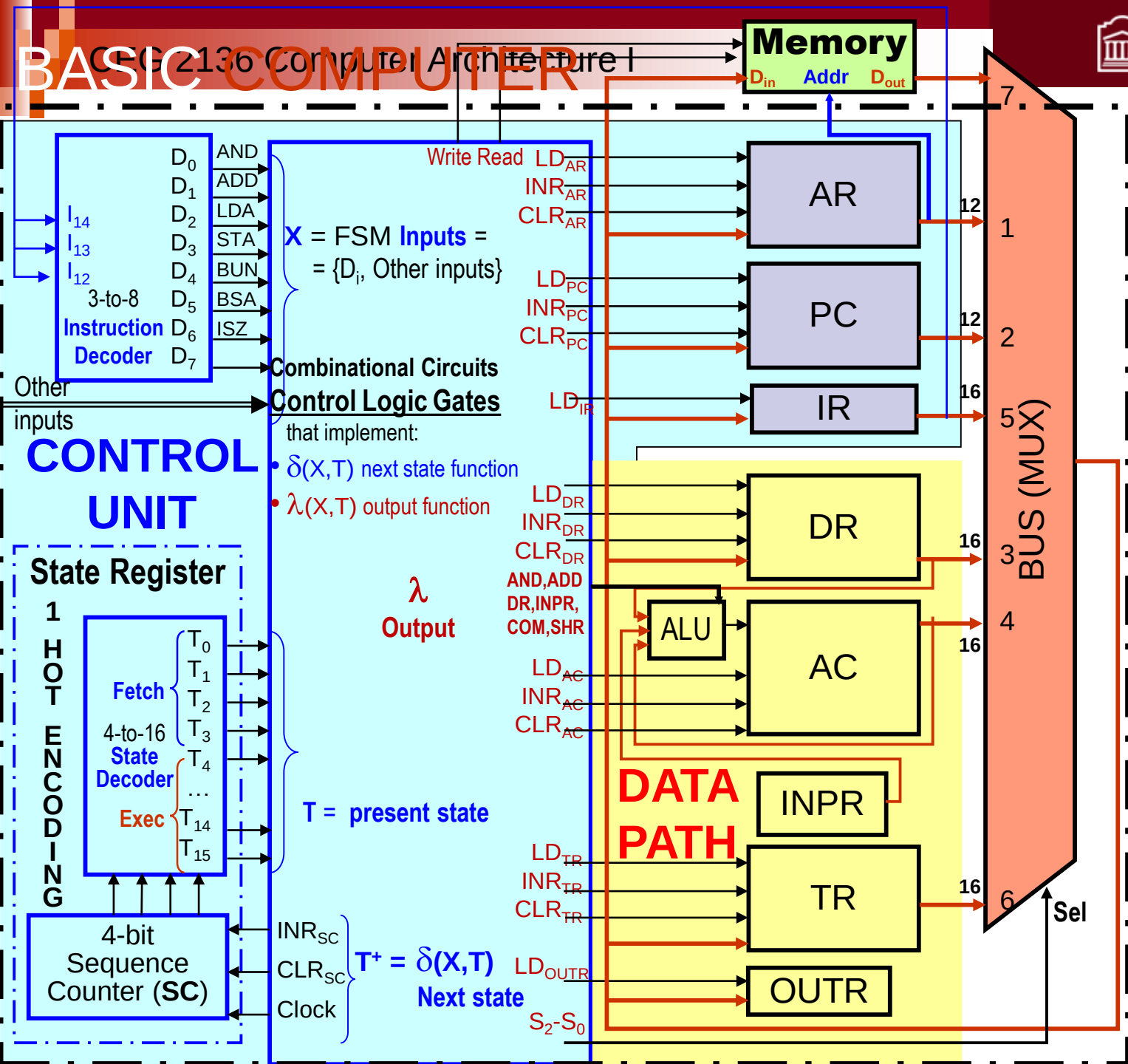


Instruction Cycle

ASM#2 Flowchart

FSM ASM#4

BASIC COMPUTER



Control Unit

The outputs of the control logic circuit in the control unit are:

1. Signals to control the inputs (LD, INR, and CLR) of the nine **registers**.
2. Signals to control the Read and Write inputs of **memory**.
3. Signals to set, clear or complement individual **flip-flops**.
4. Signals to control common **bus** selection bits: S2, S1, and S0.
5. Signals to control the **AC** adder and logic circuit.

The specifications for the various control signals can be obtained directly from the list of register transfer statements in Table 5-6 of the textbook (page 159).

Control Unit Design

Inputs: X = {D_i, Other inputs}

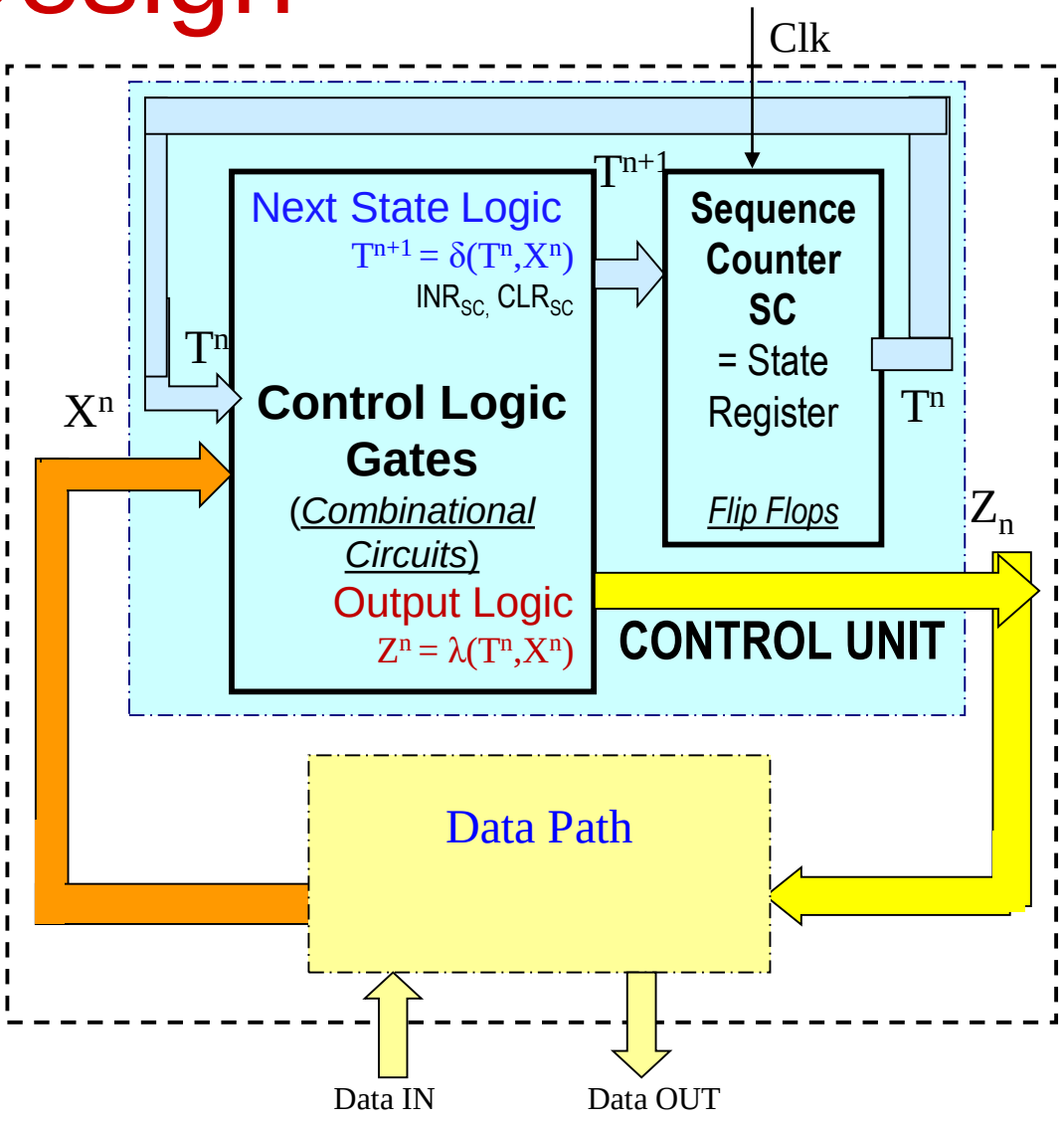
IR => D_i Instruction Decoder - AND, ADD, LDA, STA, BUN, BSA, ISZ, {RRI, IOI}

Outputs: Z = {LD_{AR}, INR_{AR}, CLR_{AR}, LD_{PC}, INR_{PC}, CLR_{PC}, Write, Read, LD_{IR}, LD_{DR}, INR_{DR}, CLR_{DR}, LD_{TR}, INR_{TR}, CLR_{TR}, LD_{AC}, INR_{AC}, …}

States : {T_0, T_1, T_2, T_3, … ,T_{15}}

$T^{n+1} = \delta(X^n,T^n)$ next state function

$Z^n = \lambda(X^n,T^n)$ output function



Registers Control Logic 1

RTL ASM#2

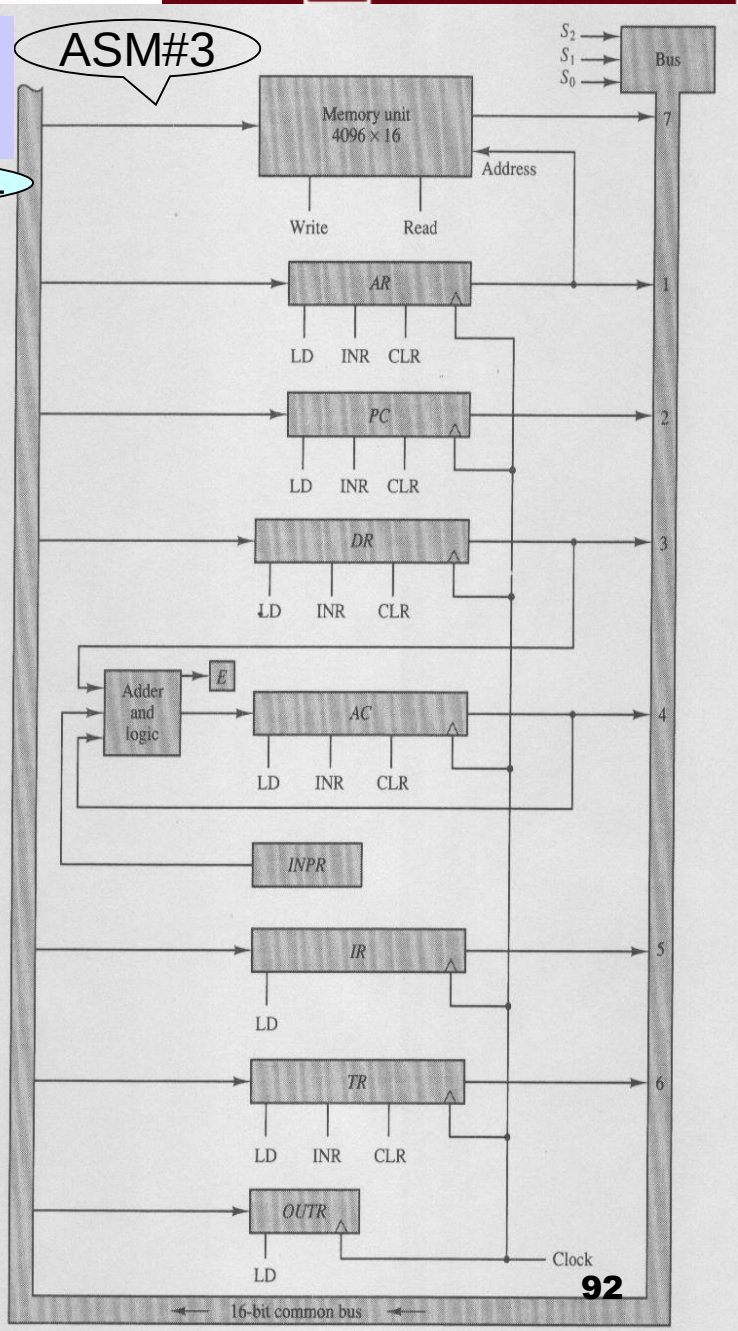
Control Signals ASM#4

			$S_2S_1S_0$	
Fetch	$R'T_0$:	$AR \leftarrow PC$	010	**LD_AR**
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$	111	Read, LD_IR, INC_PC
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode IR(12-14),		
		$AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$	101	**LD_AR**, LD_I
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$	111	Read, **LD_AR**
Interrupt:				
$T_0T_1T_2IEN(FGI + FGO)$:		$R \leftarrow 1$		S_R,
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$	010	CL_AR, LD_TR,
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$	110	Write, CL_PC
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$		INC_PC, R_{SC}, R_R, R_{IEN},
		Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$		
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$		
ADD	D_1T_4:	$DR \leftarrow M[AR]$		
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$		
LDA	D_2T_4:	$DR \leftarrow M[AR]$		
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$		
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$		
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$		
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$		
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$		
ISZ	D_6T_4:	$DR \leftarrow M[AR]$		
	D_6T_5:	$DR \leftarrow DR + 1$		
	D_6T_6:	$M[AR] \leftarrow DR$,		
	D_6T_6DR':	if(DR = 0) then ($PC \leftarrow PC + 1$), $SC \leftarrow 0$		

ASM#5

LD_AR = $R'T_0 + R'T_2 + D'_7IT_3$; CLR_AR = RT_0 ; INR_AR = D_5T_4

ASM#3



RTL_ASM#2

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$,
		$AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Interrupt:		
$T_0T_1T_2IEN(FGI + FGO)$:		$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
	Memory-reference:	
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
		.....
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$

1. $R'T_0$: $AR \leftarrow PC$
2. $R'T_2$: $AR \leftarrow IR(0\text{-}11)$
3. D'_7IT_3: $AR \leftarrow M[AR]$
4. RT_0 : $AR \leftarrow 0$
5. D_5T_4 : $AR \leftarrow AR + 1$

ASM#5

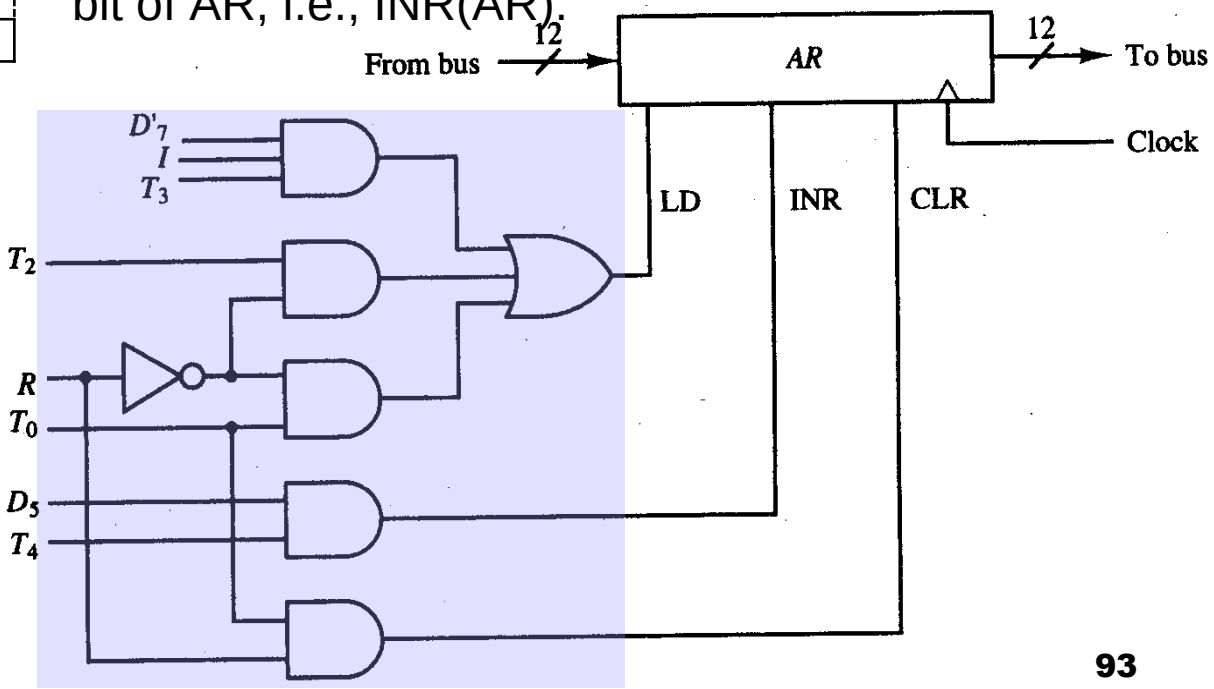
$LD(AR) = R'T_0 + R'T_2 + D'_7IT_3$

$CLR(AR) = RT_0$

$INR(AR) = D_5T_4$

AR Control Logic 1

- Scan Table 5-6 & find all the RTL statements in which AR is modified.
- The first three RTL statements are performed by enabling the LD bit of AR (i.e., LD(AR)).
- The 4-th is done by activating the CLR
- The 5-fth statement is performed by enabling the INR bit of AR, i.e., INR(AR).



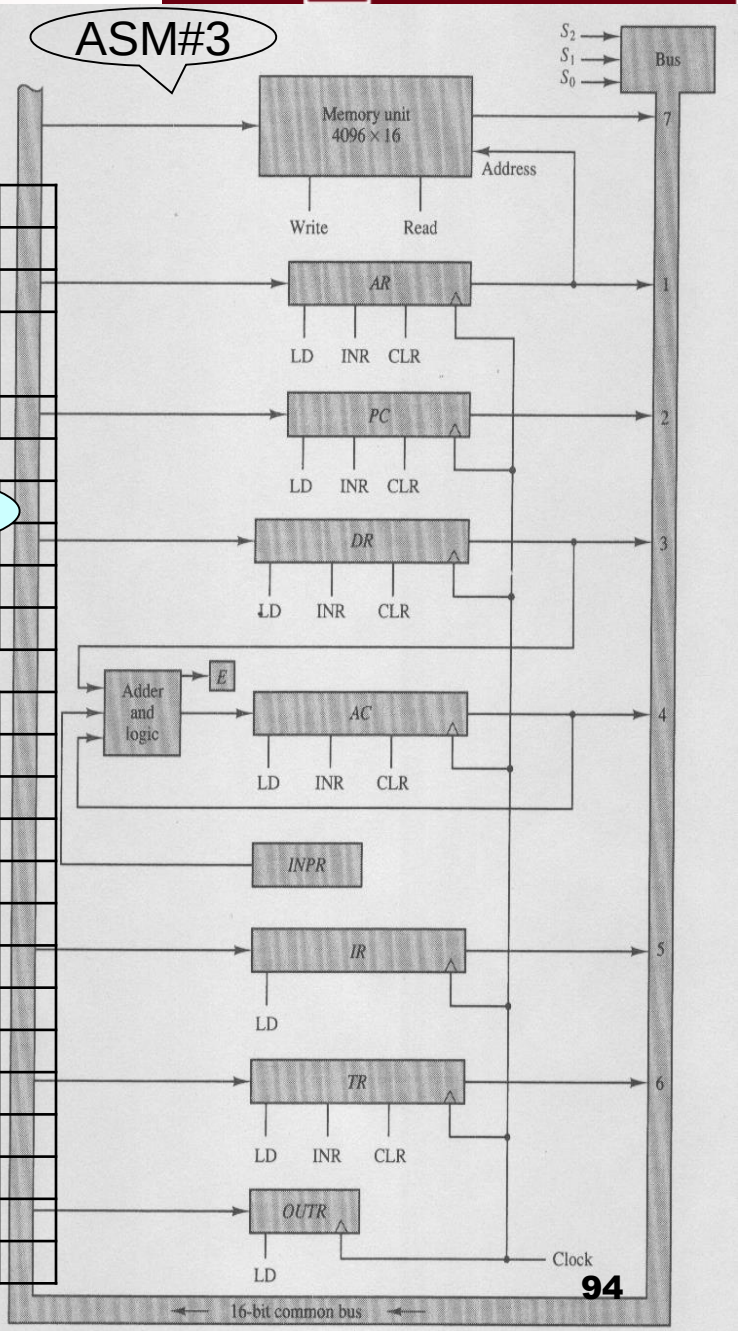
Registers Control Logic 2

RTL ASM#2

			$S_2S_1S_0$	LD_{AR}	LD_{IR}	INC_{PC}	LD_{TR}	CLR_{PC}	LD_{DR}	read
Fetch	$R'T_0$:	$AR \leftarrow PC$	010	1						
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$	111		1	1				1
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$,								
		$AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$	101	1	1					
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$	111	1						1
Interrupt:										
$T_0T_1T_2IEN(FGI + FGO)$:		$R \leftarrow 1$								
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$								
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$								
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$								
		Memory-reference:								
AND	D_0T_4:	$DR \leftarrow M[AR]$								
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$								
ADD	D_1T_4:	$DR \leftarrow M[AR]$								
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$								
LDA	D_2T_4:	$DR \leftarrow M[AR]$								
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$								
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$								
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$								
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$								
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$								
ISZ	D_6T_4:	$DR \leftarrow M[AR]$								
	D_6T_5:	$DR \leftarrow DR + 1$								
	D_6T_6:	$M[AR] \leftarrow DR$,								
	D_6T_6DR':	if $(DR = 0)$ then $(PC \leftarrow PC + 1), SC \leftarrow 0$								

Control Signals ASM#4

ASM#5



LD_AR = $R'T_0 + R'T_2 + D'_7IT_3$ ...

Control Signals ASM#4

uOttawa

Registers Control Logic

RTL ASM#2

			To	From $S_2S_1S_0$		Other / ALU	LD AR	INR AR	CLR AR	LD PC	INR PC	CLR PC	Write	Read	LD IR	LD DR	INR DR	CLR DR	LD TR	INR TR	CLR TR	LD AC	INR AC	CLR AC
Fetch	$R'T_0$:	$AR \leftarrow PC$	LD_{AR}	010			1																	
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$	LD_{IR}	111	Read	INR_{PC}					1			1	1									
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode IR(12-14),																						
		$AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$	LD_{AR}	101	LD_I		1																	
Indirect	D'_7IT_3	$AR \leftarrow M[AR]$	LD_{AR}	111	Read		1							1										
Interrupt	$T_0T_1T_2$	$IEN(FGI + FGO): R \leftarrow 1$	S_R																					
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$	LD_{TR}	010	CLR_{AR}				1															
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$	Write	110	CLR_{PC}							1												
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$	INR_{PC}	R_{IEN}	R_R	CLR_{SC}					1													
		Memory-reference:																						
AND	D_0T_4:	$DR \leftarrow M[AR]$	LD_{DR}	111	Read									1		1								
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$	LD_{AC}		CLR_{SC}	AND																1		
ADD	D_1T_4:	$DR \leftarrow M[AR]$	LD_{DR}	111	Read									1		1								
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$	LD_{AC}		CLR_{SC}	ADD																1		
LDA	D_2T_4:	$DR \leftarrow M[AR]$	LD_{DR}	111	Read									1		1								
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$	LD_{AC}		CLR_{SC}	DR																1		
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$	Write	100	CLR_{SC}																			
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$	LD_{PC}	001	CLR_{SC}					1			1											
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$	Write	010	INR_{AR}			1																
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$	LD_{PC}	001	CLR_{SC}					1			1											
ISZ	D_6T_4:	$DR \leftarrow M[AR]$	LD_{DR}	111	Read									1		1								
	D_6T_5:	$DR \leftarrow DR + 1$	INR_{DR}														1							
	D_6T_6:	$M[AR] \leftarrow DR,$	Write	011																				
	D_6T_6DR'	*if (DR = 0) then(PC <- PC + 1), SC<-0*	INR_{PC}		CLR_{SC}						1													

ASM#5

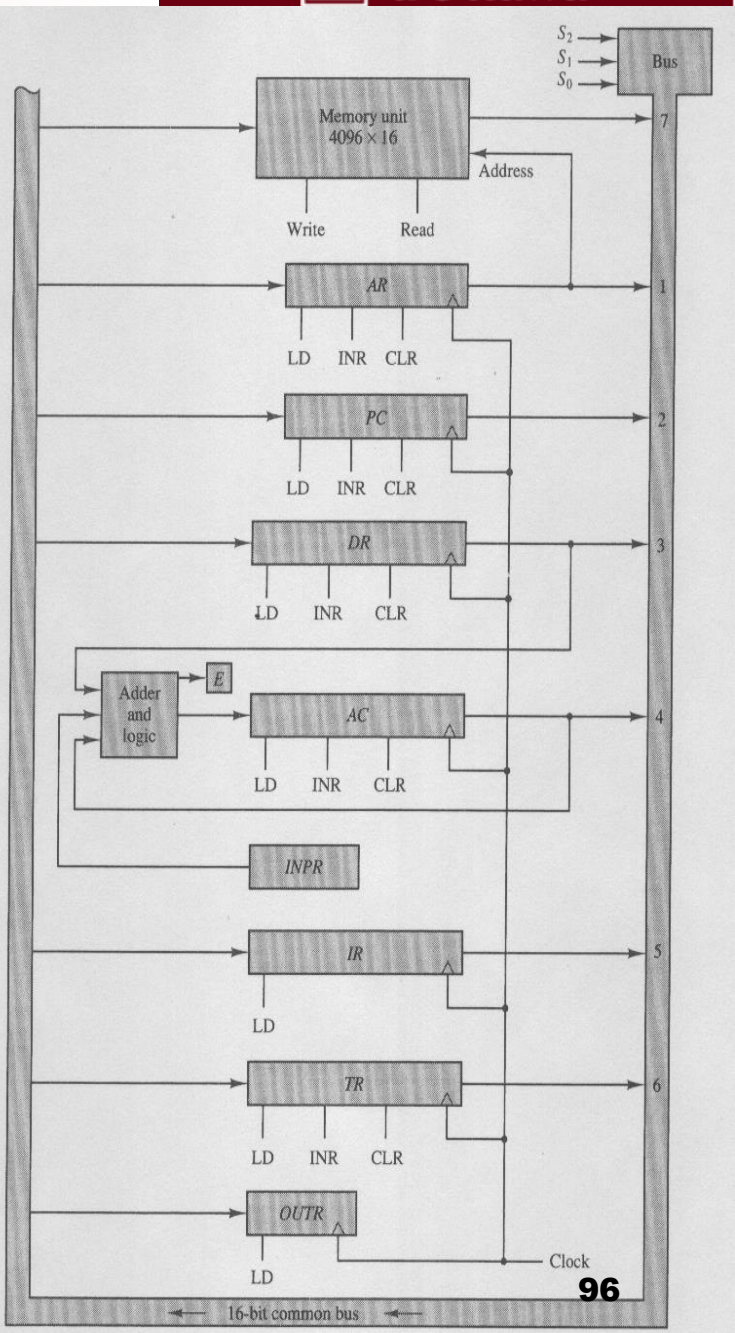
$LD\_AR = R'T_0 + R'T_2 + D'_7IT_3$; $CLR\_AR = RT_0$; $INR\_AR = D_5T_4$

FF Control Logic

		Register-reference:	R_{IEN}	S_{IEN}				
	$D_7I'T_3 = r$	(common to all RRI)						
	$IR(i) = B_i$	(i = 0, 1, 2, ... , 11)						
	r:	SC <- 0						
CLA	rB_{11}:	*AC <- 0*						
CLE	rB_{10}:	*E <- 0*						
CMA	rB_9:	*AC <- AC'*						
CME	rB_8:	*E <- E'*						
CIR	rB_7:	*AC <- shr AC, AC(15) <- E, E <- AC(0)*						
CIL	rB_6:	*AC <- shlAC, AC(0) <- E, E <- AC(15)*						
INC	rB_5:	*AC <- AC + 1*						
SPA	$rB_4AC'(15)$	*If (AC(15) = 0) then (PC <- PC + 1)*						
SNA	rB_3AC(15):	*If (AC(15) = 1) then (PC <- PC + 1)*						
SZA	rB_2AC':	*If (AC = 0) then (PC <- PC + 1)*						
SZE	rB_1E':	*If (E = 0) then (PC <- PC + 1)*						
HLT	rB_0:	*S <- 0*						
		Input-output:						
	$D_7IT_3 = p$	(common to all IO instructions)						
	$IR(i) = B_i$	(i = 6, 7, 8, 9, 10, 11)						
	p:	*SC <- 0*						
INP	pB_{11}:	*AC(0-7) <- INPR, FGI <- 0*						
OUT	pB_{10}:	*OUTR <- AC(0-7), FGO <- 0*						
SKI	pB_9FGI:	*If (FGI = 1) then (PC <- PC + 1)*						
SKO	pB_8FGO:	*If (FGO = 1) then (PC <- PC + 1)*						
ION	pB_7:	***IEN <- 1***		1				
IOF	pB_6:	***IEN <- 0***	1					

Control Signals ASM#4

RTL ASM#2



IEN FF Control Logic

- The control logic circuit for the individual flip-flops of the control unit can be determined in a similar manner.
- For example, Table 5-6 (of the textbook) shows that flip-flop IEN is loaded in the following statements:

RTL ASM#2

- pB_7: $IEN \leftarrow 1$ $S_{IEN} = 1$
- pB_6: $IEN \leftarrow 0$ $R_{IEN1} = 1$
- RT_2: $IEN \leftarrow 0$ $R_{IEN2} = 1$ where $p = D_7 I T_3$, $B_7 = IR(7)$, and $B_6 = IR(6)$

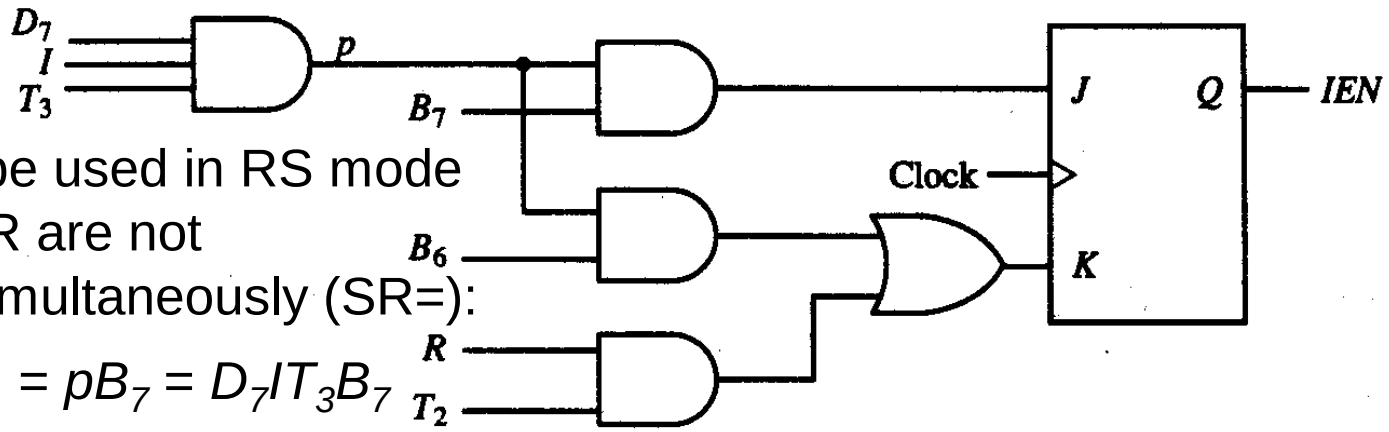
Control Signals ASM#4

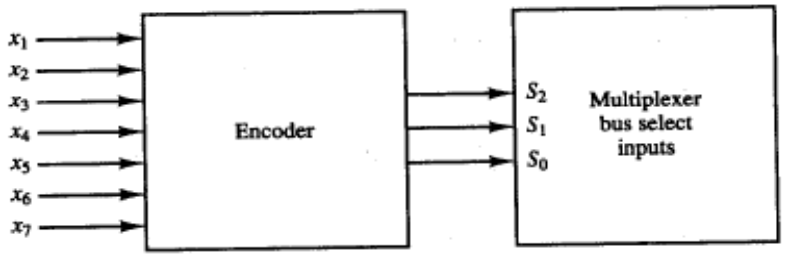
ASM#5

A JK flip-flop can be used in RS mode since here S and R are not required to be 1 simultaneously (SR=):

$$J_{IEN} = \Sigma(S_{IEN}) = pB_7 = D_7 I T_3 B_7$$

$$K_{IEN} = \Sigma(R_{IEN}) = pB_6 + RT_2$$





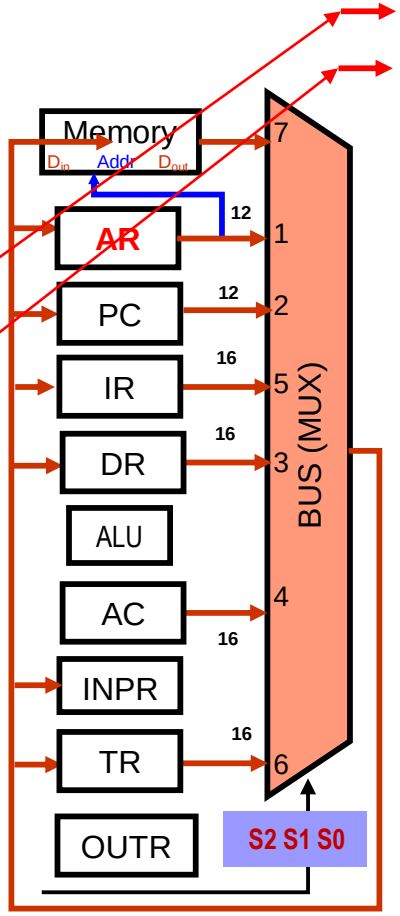
Bus Control

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0
AR	1	0	0	0	0	0	0	0	0	1
PC	0	1	0	0	0	0	0	0	1	0
DR	0	0	1	0	0	0	0	0	1	1
AC	0	0	0	1	0	0	0	1	0	0
IR	0	0	0	0	1	0	0	1	0	1
TR	0	0	0	0	0	1	0	1	1	0
M	0	0	0	0	0	0	1	1	1	1

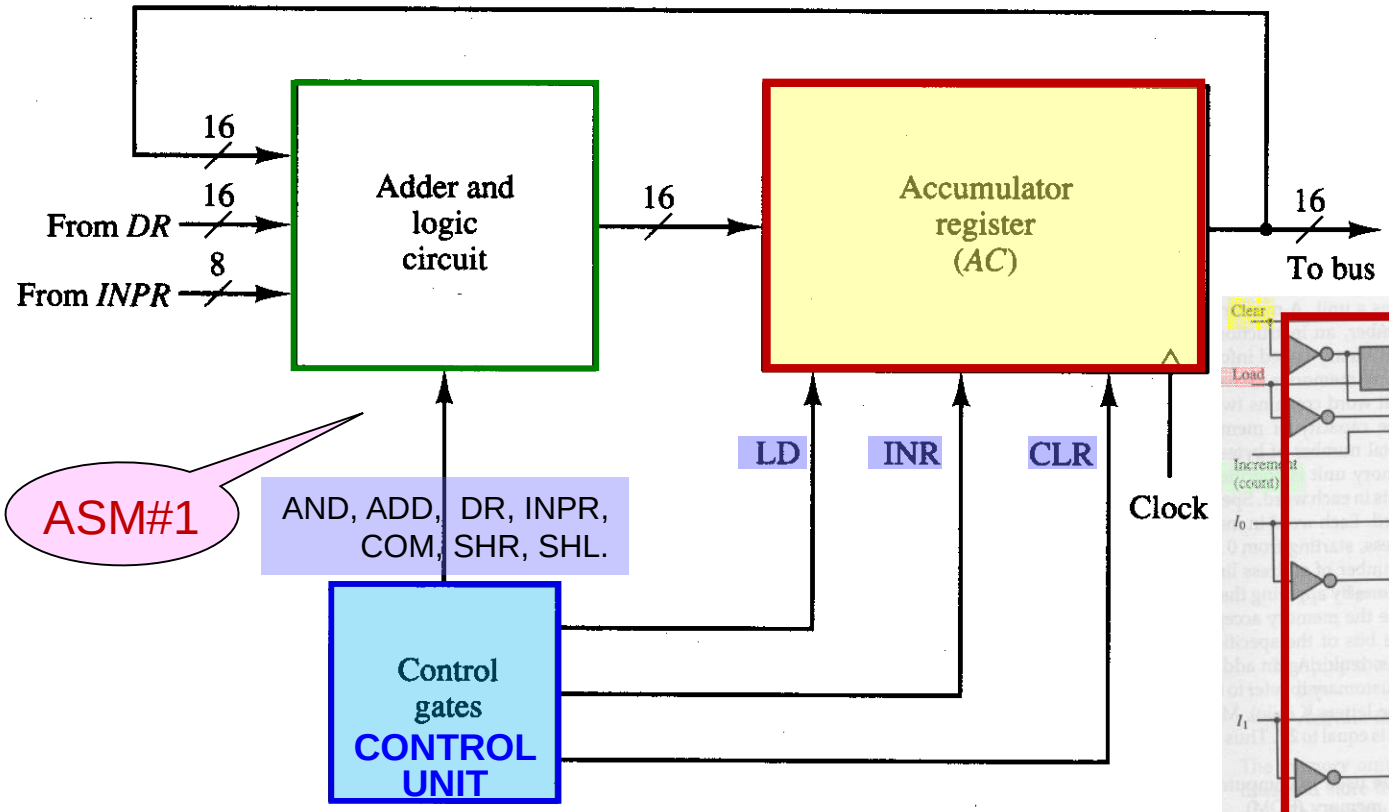
$S_0 = x_1 + x_3 + x_5 + x_7$
$S_1 = x_2 + x_3 + x_6 + x_7$
$S_2 = x_4 + x_5 + x_6 + x_7$

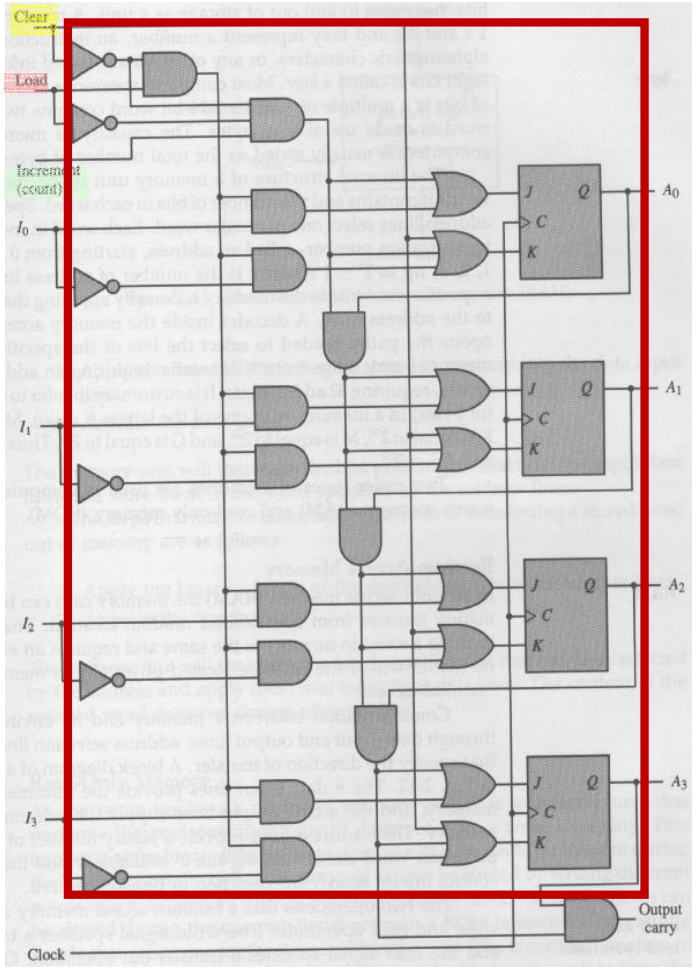
To find x_1 scan the table of microoperations of Basic Computer from Table 5-6 and collect all the RTL statements where **AR** is a **Source Register** to be selected by the Bus MUX (on the *right side* of RTL!)

AR:	***D_4T_4: PC <- AR*** ***D_5T_5: PC <- AR***	Sel_{AR}	$= x_1 = D_4T_4 + D_5T_5$
PC:		Sel_{PC}	$= x_2 =$
DR:		Sel_{DR}	$= x_3 =$
AC:		Sel_{AC}	$= x_4 =$
IR :		Sel_{IR}	$= x_5 =$
TR:		Sel_{TR}	$= x_6 =$
M :		Sel_M	$= x_7 = R'T_1 + D'_7IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$



Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR],\ PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$,
		$AR \leftarrow IR(0\text{-}11),\ I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Interrupt	$IEN\ (FGI+FGO)$:	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0,\ TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR,\ PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC+1,\ IEN \leftarrow 0,\ R \leftarrow 0,\ SC \leftarrow 0$
Execute Memory-reference Instructions (MRI)		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR,\ SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR,\ E \leftarrow C_{out},\ SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR,\ SC \leftarrow 0$
STA.	D_3T_4:	$M[AR] \leftarrow AC,\ SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR,\ SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC,\ AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR,\ SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR,$
	D_6T_6DR':	$if\ (DR = 0)\ then\ (PC \leftarrow PC + 1),\ SC \leftarrow 0$
Execute Register-reference Instructions (RRI)		
	$D_7I'T_3 = r$	(common to all Register-ref. instructions)
		$IR(i) = B_i$ (i = 0, 1, 2, ... , 11)
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	$rB_4AC'(15)$:	$If\ (AC(15) = 0)\ then\ (PC \leftarrow PC + 1)$
SNA	$rB_3AC(15)$:	$If\ (AC(15) = 1)\ then\ (PC \leftarrow PC + 1)$
SZA	rB_2AC':	$If\ (AC = 0)\ then\ (PC \leftarrow PC + 1)$
SZE	rB_1E':	$If\ (E = 0)\ then\ (PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
Execute Input-output (IOI)		
	$D_7IT_3 = p$	(common to all input-output instructions)
	$IR(i) = B_i$	(i = 6, 7, 8, 9, 10, 11)
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR,\ FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7),\ FGO \leftarrow 0$
SKI	pB_9FGI:	$If\ (FGI = 1)\ then\ (PC \leftarrow PC + 1)$
SKO	pB_8FGO:	$If\ (FGO = 1)\ then\ (PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

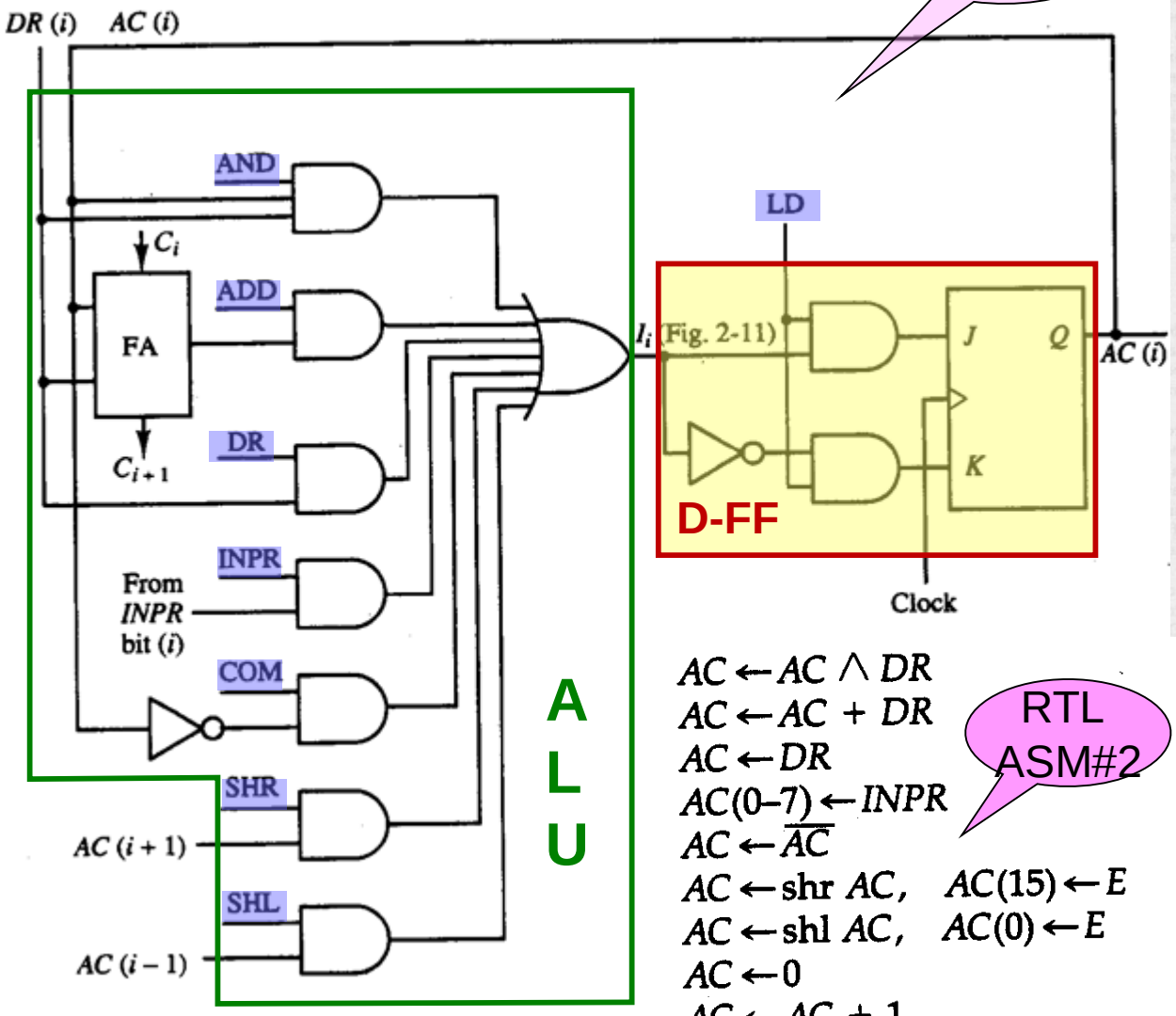


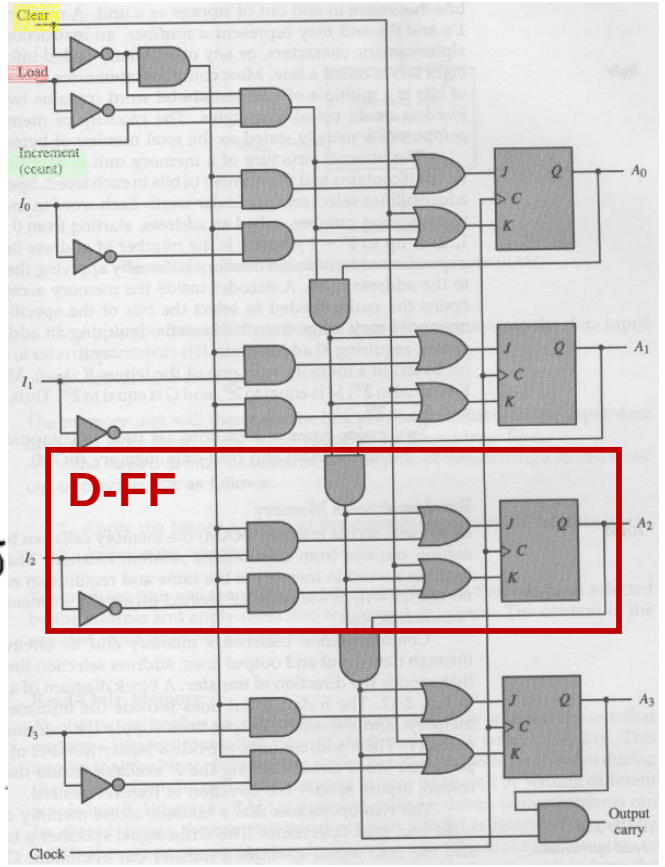


ALU Block Diagram

AC+ALU

ASM#3





RTL ASM#2

$AC \leftarrow AC \wedge DR$		AND with *DR*
$AC \leftarrow AC + DR$		Add with *DR*
$AC \leftarrow DR$		Transfer from *DR*
$AC(0\text{–}7) \leftarrow INPR$		Transfer from *INPR*
$AC \leftarrow \overline{AC}$		Complement
$AC \leftarrow \text{shr } AC,$	$AC(15) \leftarrow E$	Shift right
$AC \leftarrow \text{shl } AC,$	$AC(0) \leftarrow E$	Shift left
$AC \leftarrow 0$		Clear
$AC \leftarrow AC + 1$		Increment

ALU Control

To design the logic associated with AC, it is necessary to scan Table 5-6 (of the textbook) and extract all the statements in which AC is loaded:

RTL ASM#2

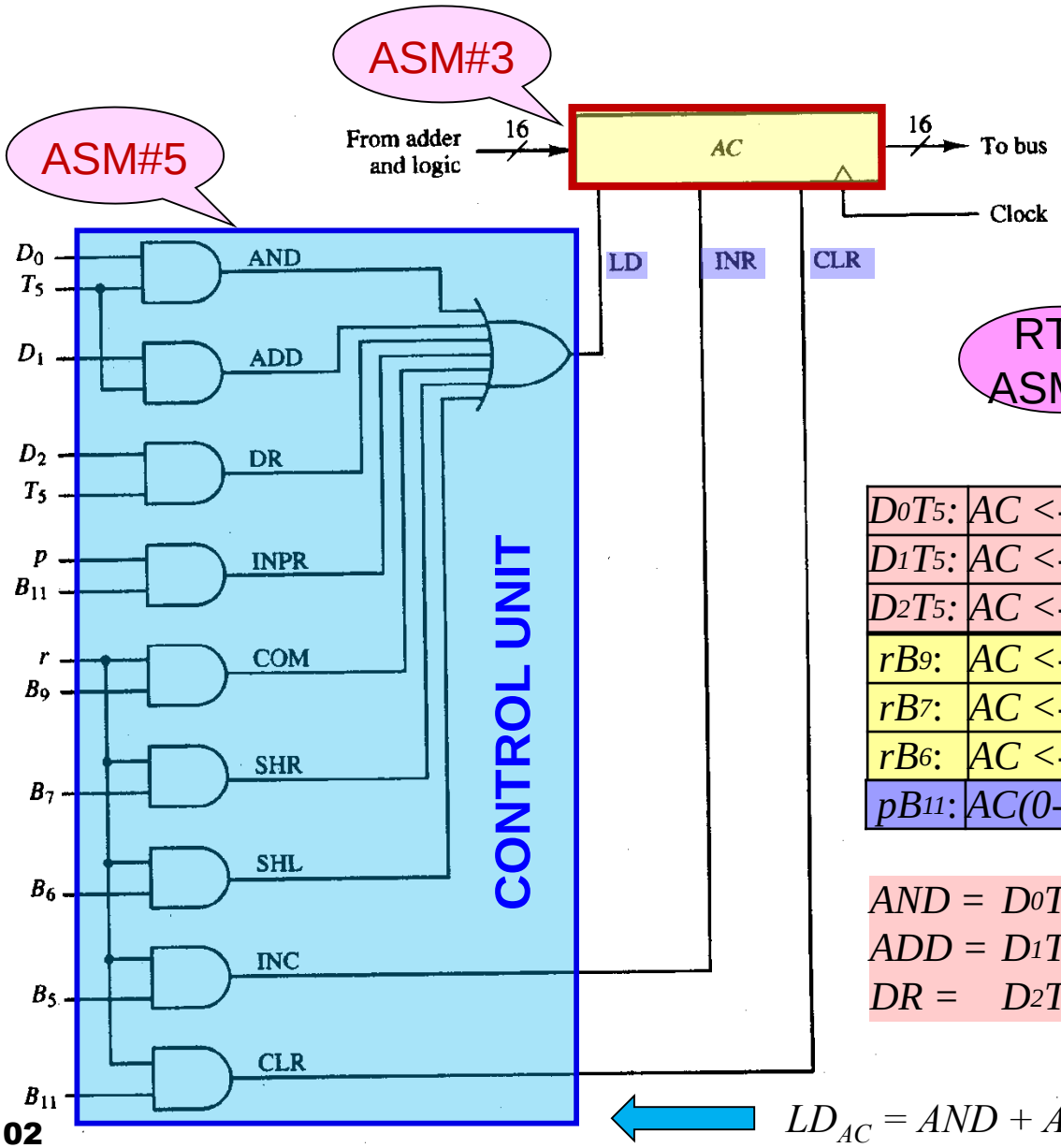
$AC \leftarrow AC \wedge DR$	AND with *DR*
$AC \leftarrow AC + DR$	Add with *DR*
$AC \leftarrow DR$	Transfer from *DR*
$AC(0\text{–}7) \leftarrow INPR$	Transfer from *INPR*
$AC \leftarrow \overline{AC}$	Complement
$AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E$	Shift right
$AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E$	Shift left
$AC \leftarrow 0$	Clear
$AC \leftarrow AC + 1$	Increment

ASM#4

D_0T_5:	$AC \leftarrow AC \wedge DR,\ SC \leftarrow 0$	*AND*
D_1T_5:	$AC \leftarrow AC + DR,\ E \leftarrow C_{out},\ SC \leftarrow 0$	*ADD*
D_2T_5:	$AC \leftarrow DR,\ SC \leftarrow 0$	*DR*
rB_9:	$AC \leftarrow AC'$	*COM*
rB_7:	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	*SHR*
rB_6:	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	*SHL*
pB_{11}:	$AC(0\text{-}7) \leftarrow INPR,\ FGI \leftarrow 0$	*INPR*

			ALU	*AC*
Fetch	$R'T_0$:	$AR \leftarrow PC$		
	$R'T_1$:	$IR \leftarrow M[AR],\ PC \leftarrow PC + 1$		
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$,		
		$AR \leftarrow IR(0\text{-}11),\ I \leftarrow IR(15)$		
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$		
Interrupt	$IEN\ (FGI+FGO)$: $R \leftarrow 1$			
	RT_0:	$AR \leftarrow 0,\ TR \leftarrow PC$		
	RT_1:	$M[AR] \leftarrow TR,\ PC \leftarrow 0$		
	RT_2:	$PC \leftarrow PC+1,\ IEN \leftarrow 0,\ R \leftarrow 0,\ SC \leftarrow 0$		
Execute	**Memory-reference Instructions (MRI)**			
AND	D_0T_4:	$DR \leftarrow M[AR]$		
	D_0T_5:	$AC \leftarrow AC \wedge DR,\ SC \leftarrow 0$	*AND*	LDAC
ADD	D_1T_4:	$DR \leftarrow M[AR]$		
	D_1T_5:	$AC \leftarrow AC + DR,\ E \leftarrow C_{out},\ SC \leftarrow 0$	*ADD*	LDAC
LDA	D_2T_4:	$DR \leftarrow M[AR]$		
	D_2T_5:	$AC \leftarrow DR,\ SC \leftarrow 0$	*DR*	LDAC
STA.	D_3T_4:	$M[AR] \leftarrow AC,\ SC \leftarrow 0$		
BUN	D_4T_4:	$PC \leftarrow AR,\ SC \leftarrow 0$		
BSA	D_5T_4:	$M[AR] \leftarrow PC,\ AR \leftarrow AR + 1$		
	D_5T_5:	$PC \leftarrow AR,\ SC \leftarrow 0$		
ISZ	D_6T_4:	$DR \leftarrow M[AR]$		
	D_6T_5:	$DR \leftarrow DR + 1$		
	D_6T_6:	$M[AR] \leftarrow DR,$		
	D_6T_6DR':	*if* $(DR = 0)$ *then* $(PC \leftarrow PC + 1),\ SC \leftarrow 0$		
Execute	**Register-reference Instructions (RRI)**			
	$D_7I'T_3 = r$	(common to all Register-ref. instructions)		
		$IR(i) = B_i$ (i = 0, 1, 2, ... , 11)		
	r:	$SC \leftarrow 0$		
CLA	rB_{11}:	$AC \leftarrow 0$		CLRAC
CLE	rB_{10}:	$E \leftarrow 0$		
CMA	rB_9:	$AC \leftarrow AC'$	*COM*	LDAC
CME	rB_8:	$E \leftarrow E'$		
CIR	rB_7:	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	*SHR*	LDAC
CIL	rB_6:	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	*SHL*	LDAC
INC	rB_5:	$AC \leftarrow AC + 1$		INCAC
SPA	$rB_4AC'(15)$:	*If* $(AC(15) = 0)$ *then* $(PC \leftarrow PC + 1)$		
SNA	$rB_3AC(15)$:	*If* $(AC(15) = 1)$ *then* $(PC \leftarrow PC + 1)$		
SZA	rB_2AC':	*If* $(AC = 0)$ *then* $(PC \leftarrow PC + 1)$		
SZE	rB_1E':	*If* $(E = 0)$ *then* $(PC \leftarrow PC + 1)$		
HLT	rB_0:	$S \leftarrow 0$		
Execute	**Input-output (IOI)**			
	$D_7IT_3 = p$	(common to all input-output instructions)		
	$IR(i) = B_i$	(i = 6, 7, 8, 9, 10, 11)		
	p:	$SC \leftarrow 0$		
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR,\ FGI \leftarrow 0$	*INPR*	LDAC
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7),\ FGO \leftarrow 0$		
SKI	pB_9FGI:	*If* $(FGI = 1)$ *then* $(PC \leftarrow PC + 1)$		
SKO	pB_8FGO:	*If* $(FGO = 1)$ *then* $(PC \leftarrow PC + 1)$		
ION	pB_7:	$IEN \leftarrow 1$		
IOF	pB_6:	$IEN \leftarrow 0$		

AC + ALU Control Logic (ASM#5)



RTL ASM#2 / Signals ASM#4

D_0T_5:	AC <- AC ^ DR, SC <- 0	AND
D_1T_5:	AC <- AC + DR, E <- Cout, SC <- 0	ADD
D_2T_5:	AC <- DR, SC <- 0	DR
rB_9:	AC <- AC'	COM
rB_7:	AC <- shr AC, AC(15) <- E, E <- AC(0)	SHR
rB_6:	AC <- shl AC, AC(0) <- E, E <- AC(15)	SHL
pB_{11}:	AC(0-7) <- INPR, FGI <- 0	INPR

$AND = D_0T_5$
$ADD = D_1T_5$
$DR = D_2T_5$

$COM = rB_9$
$SHR = rB_7$
$SHL = rB_6$

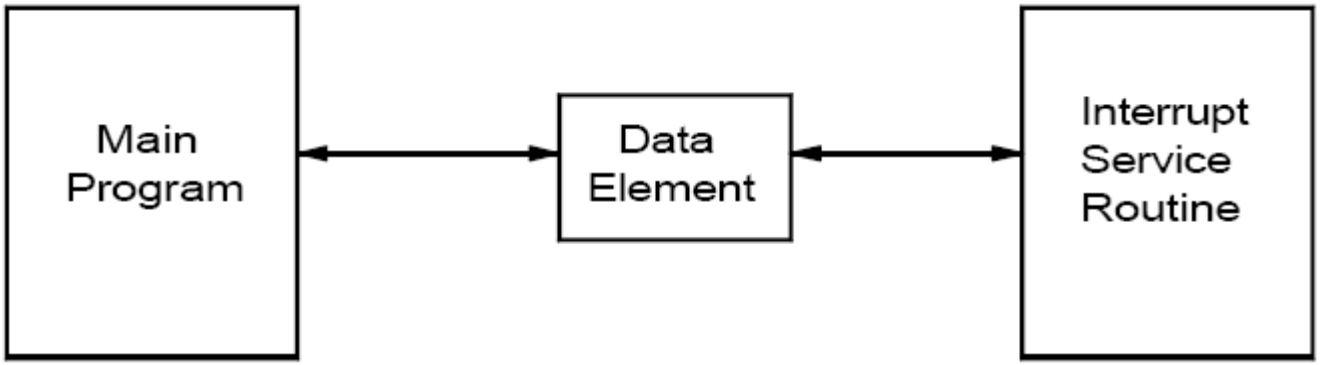
$INPR = pB_{11}$

$$LD_{AC} = AND + ADD + DR + COM + SHR + SHL + INPR$$

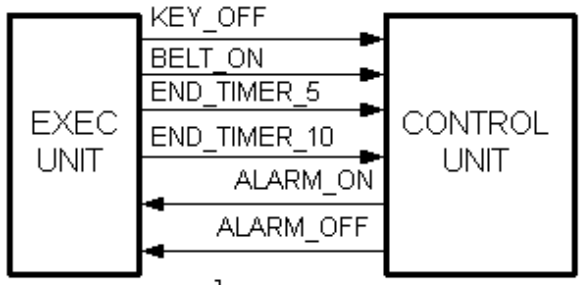
Data Exchange with ISRs

- Use global data
- May need to disable interrupts for critical regions of code using this data

Input-Output and Interrupt



ASM Example




```
OFF
        if (KEY_ON)
                START_TIMER
                goto WAIT
        else (goto OFF)
WAIT
        if (KEY_OFF or BELT_ON)
                if (END_TIMER_5)
                        ALARM_ON
                else goto WAIT
        else goto ALARM
ALARM
        if (BELT_ON  or END_TIMER_10 or KEY_OFF)
                ALARM_OFF
                goto OFF
        else goto ALARM
```