COE 608

Computer Organization & Architecture

Course Sum-Up Notes

Winter 2012 Semester 6

Table of Contents

Instruction Set Design and Architecture (ISA)

The instruction architecture must deal with the following states:

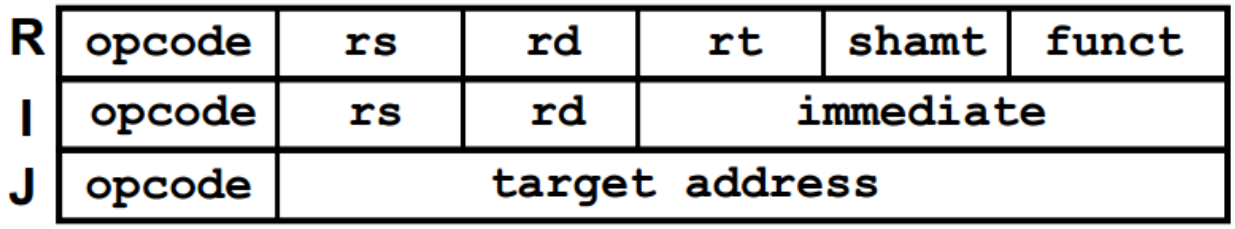
Instruction Fetch → Instruction Decode →Operand Fetch → Execute → Result Store → Next Instruction

MIPS-Processor Instruction Registers:

Name	Register number	Usage
$zero	0	the constant value 0
$v0-$v1	2-3	values for results and expression evaluation
$a0-$a3	4-7	arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved
$t8-$t9	24-25	more temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

The MIPS CPU comes with these 32 CPU registers. The CPU designed in the class is much more simple.

The 3 MIPS Instruction Formats



R-Format: Used for all other instructions.

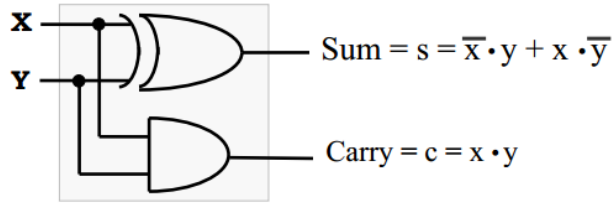
I-Format: Used for instructions with immediate values

J-Format: Used for Jump instructions

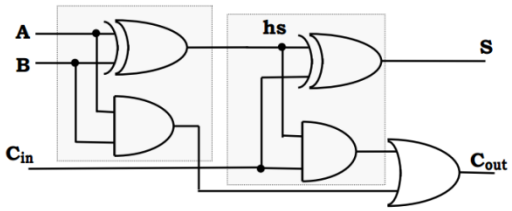
Important Notes: For J-format instructions the target address is 26 bits and all address values are stores as words; therefore an extra 00 is needed.

Adder Circuits:

1-Bit Half-Adder



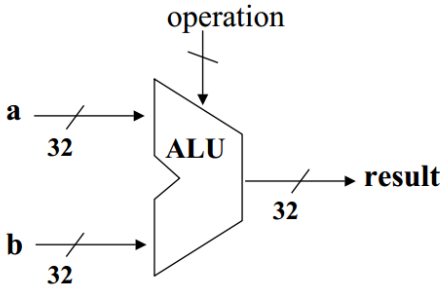
1-Bit Full-Adder



To Subtract we just simply used 2's compliment addition: X – Y = X + !Y + 1

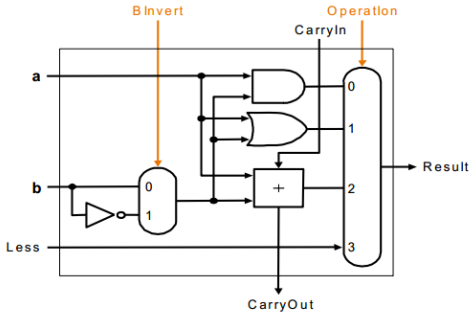
ALU Architecture:

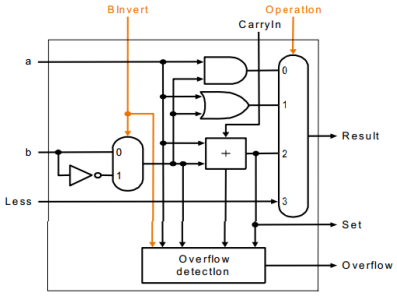
The ALU is responsible for all arithmetic and logical operations in the CPU. In lab 3a and 3b a 32/8bit CPU was implemented and designed using VHDL.



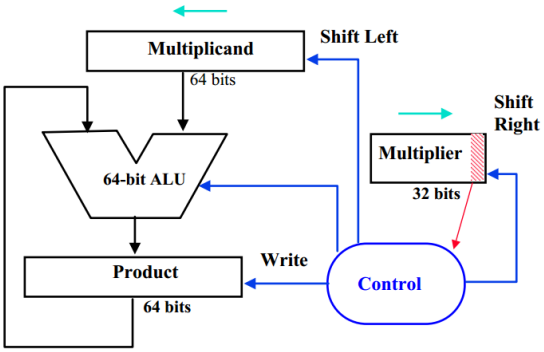
See lab 3 for more details on the ALU and its operations

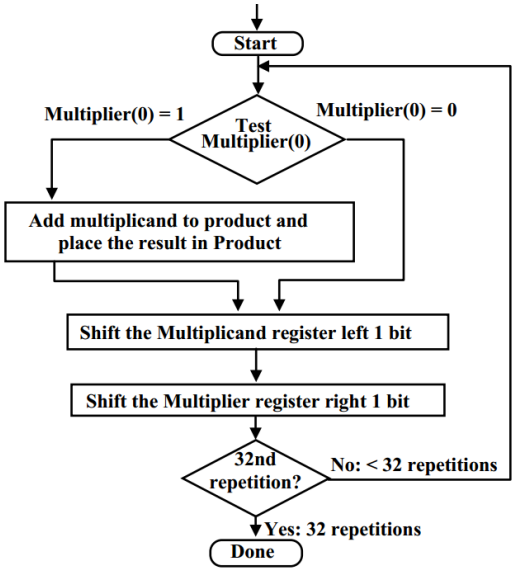
Example of **SLT** Instruction support by the MIPS ALU: The 2nd figure is the same circuit with overflow detection.





Unsigned Shift-Add Multiplier: Techniques for multiplying





Multiplier = datapath + control

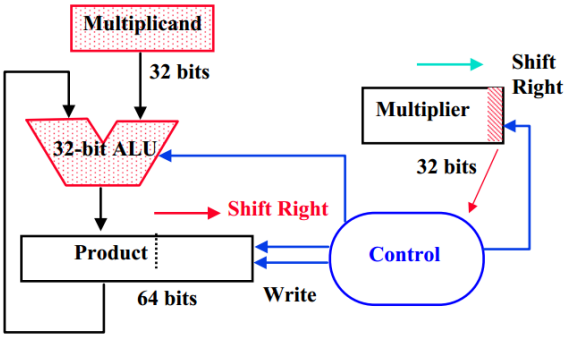
Using the following state diagram on the right, a table can be formed and the unsigned shift-add multiplier will give the resultant product.

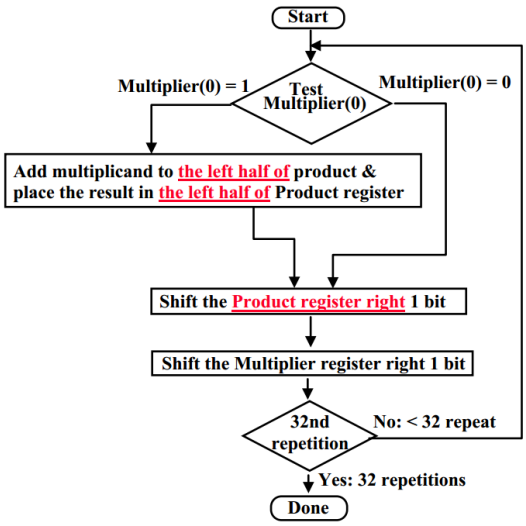
Good questions to review for Exam and method for replacing mul q.

Product	Multiplier	Multiplicand
6	**3**	**2**
0000 0000	0011	0000 0010
0000 0010	1001	0000 0100
0000 0110	**1100**	**0000 1000**

Another Multiplier Algorithm:

Multiplier	Multiplicand	Product
0011	0010	0000 0000

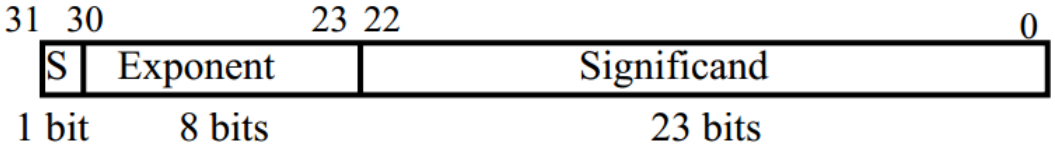




Floating Point Arithmetic:

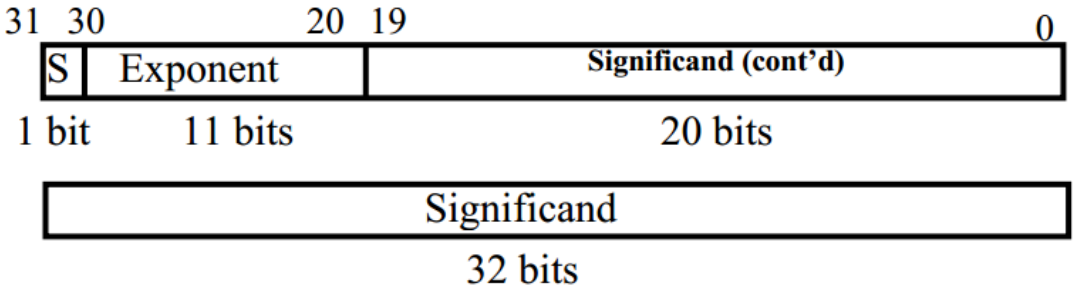
Single Precision

Normal Format: $+1.xxxxxxxxxx_{two} \times 2^{yyyy_{two}}$



$$(-1)^{S} \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$$

Double Precision



Double precision exponent is +1023 instead of +127. It also can store 52 bits as its significand.

Computer Performance

$$t = \frac{\sum I * CPI}{clk}, where\ t\ is\ time, I\ is\ intructions, CPI\ cycles\ per\ intruction\ and\ clk\ is\ clock$$

$$Number\ of\ cycles = CPI * I$$

$$IPC = \frac{1}{CPI}$$

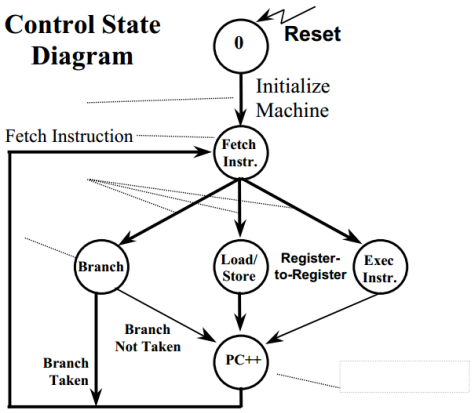
$$Global\ CPI = \frac{t * clk}{I}, where\ t\ is\ found\ from\ first\ equation$$

$$Speedup = \frac{Exectution\ Time\ Before\ Enhancement}{Execution\ Time\ After\ Enhancment}\ in\ \%$$

Best way to do these questions is just review the ones given in the midterm.

Register Transfer and Data-path

The data-path is the flow of all information within the CPU. By designing it properly and creating fast structure, information can easily and quickly flow throughout the CPU. The data-path is controlled by a Control Unit that used state diagrams to send information.



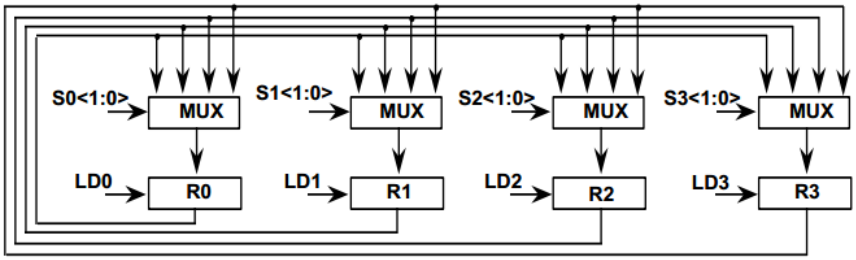
Register transfers is the transfer of information from one register to another using a data-path. The control unit generates control signals and from this registers transfer information.

Micro-Operations

Micro-operations are the elementary operations performed on registers. They are usually preformed in one clock cycle; multiple independent micro-ops can be performed in one clock cycle as well. One for each control point.

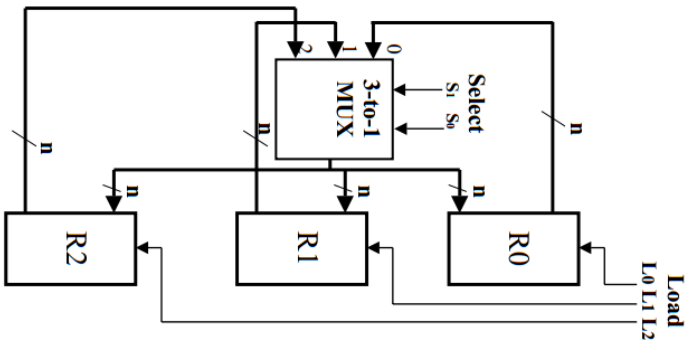
Data-path Interconnection Strategies:

Point-to-Point Connection

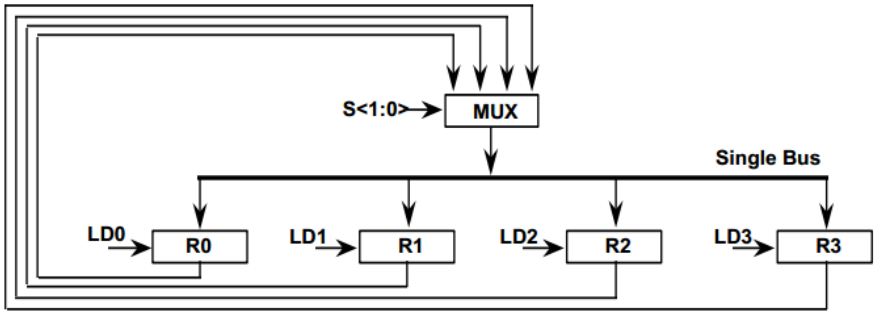


Midterm Q related to this topic. First find all sources and destinations then design to simplify the circuits.

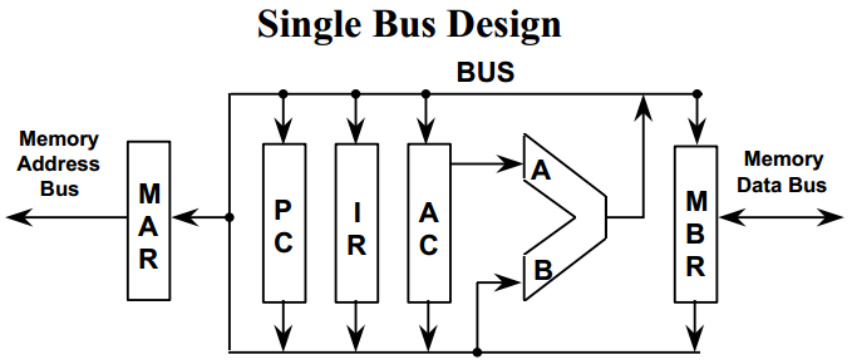
MUS and Bus Transfer



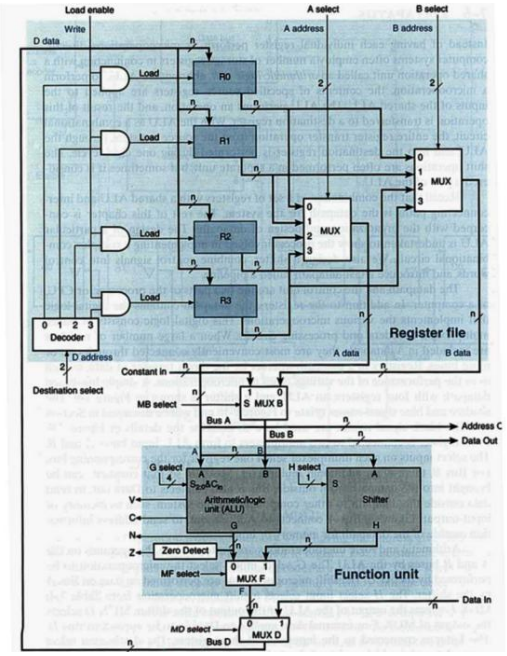
Single Bus Interconnection



Bus-based Interconnection



General Data-path



Single Cycle MIPS Lite Data-path Design

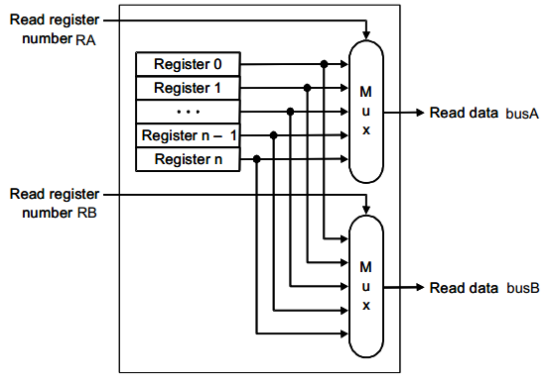
MIPS instruction formats are 32 bits long and support: op, funct, rs, rt, rd, shamt, address/immediate, target address.

The basic building blocks include an adder, mux and ALU.

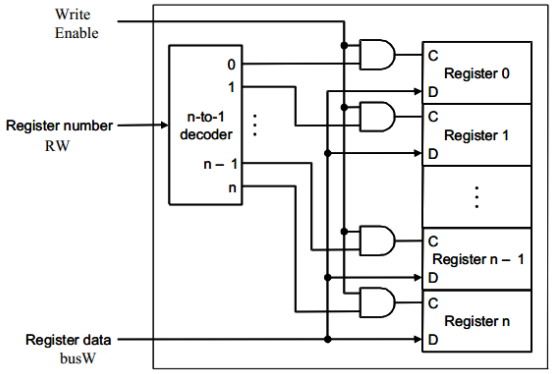
Register File:

Reg Files are a much cleaner way to store all information registers.

Reading:



Writing:

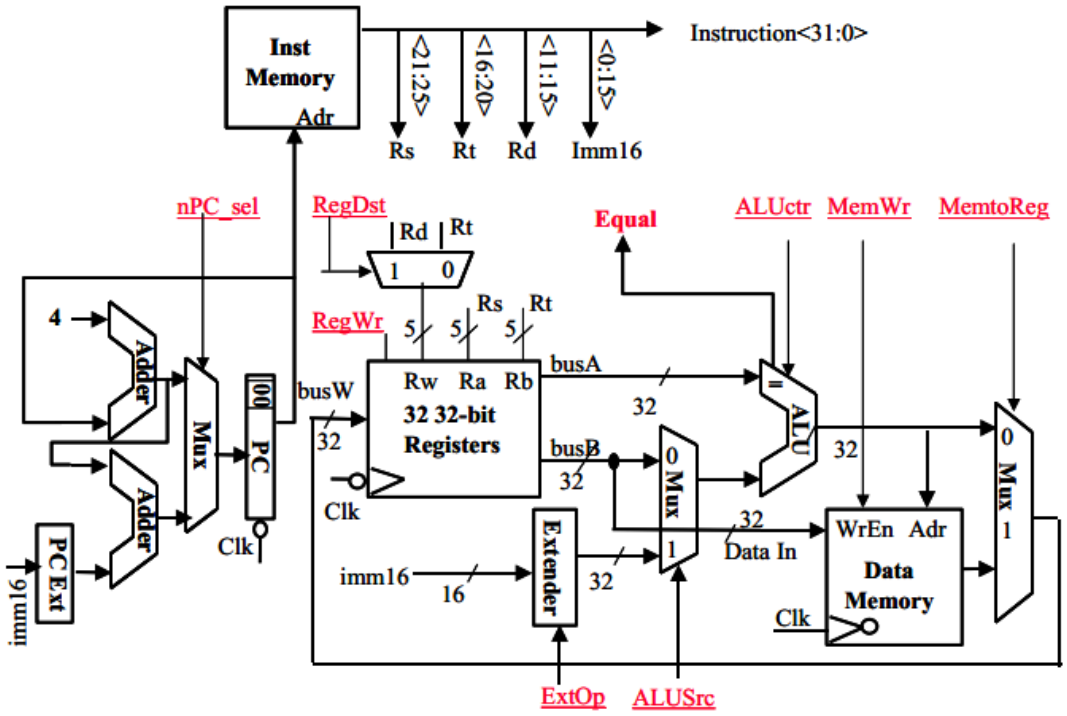


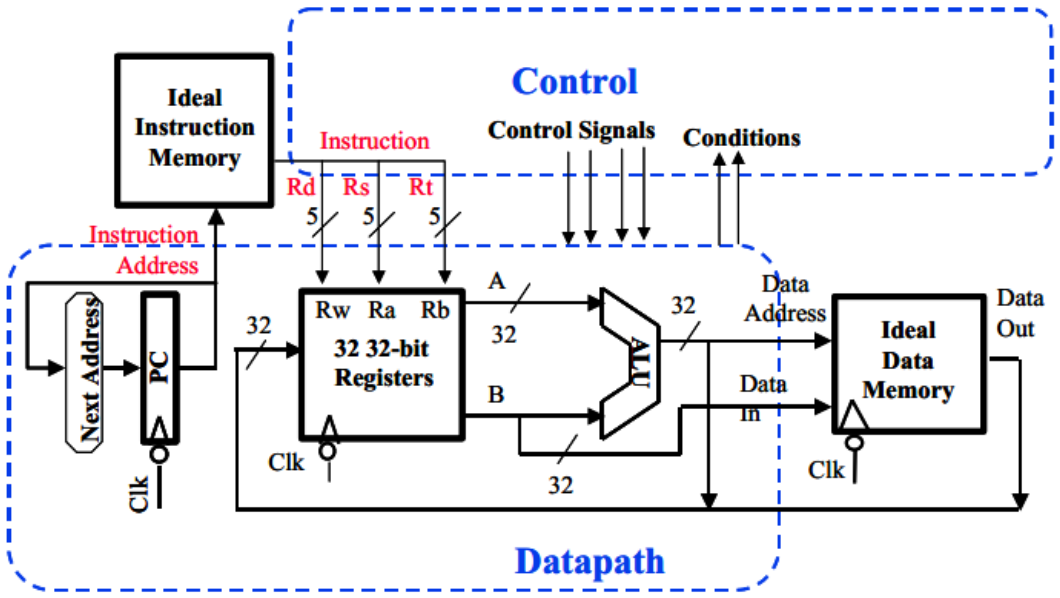
Clock Cycle:

The clock will have an edge triggered methodology. All storage elements are clocked by the same clock edge. The cycle time can be calculated by:

Cycle Time = CLK-to-Q + Max-Delay-Path + Setup + Clock Skew

Putting it all together: The final data-path for single cycle

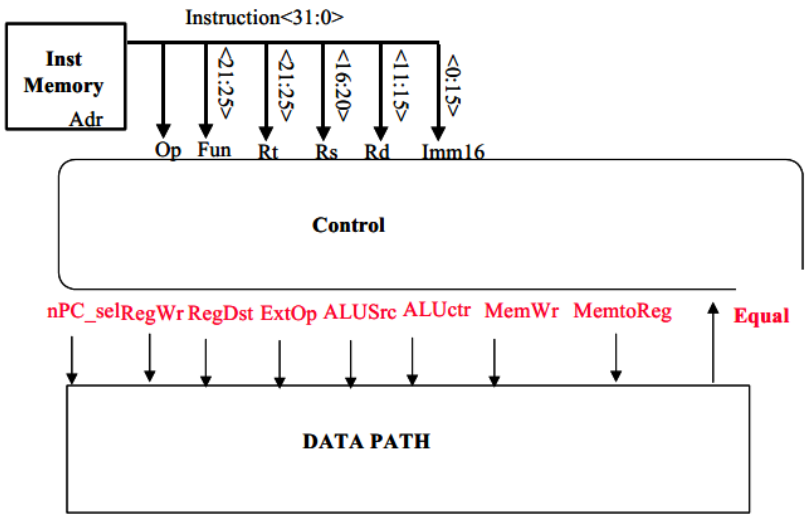




MIPS-Lite Single-Cycle Control

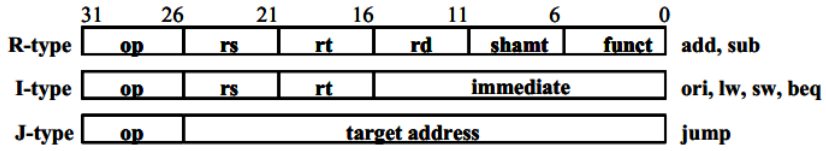
5 Steps to design a processor:

1) Analyze instruction set => data path requirements
2) Select the set of data path components and establish clock methodology
3) Assemble data path meeting the requirements
4) Analyze implementation of each instruction to determine setting of control points that affects the register transfer
5) Assemble the control logic



See → func	10 0000	10 0010	Don't Care				
See → op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	sub	ori	lw	sw	beq	jump
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWr	1	1	1	1	0	0	0
MemWr	0	0	0	0	1	0	0
nPCsel	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract	xxx

There is a method to encode and decode all the signals.



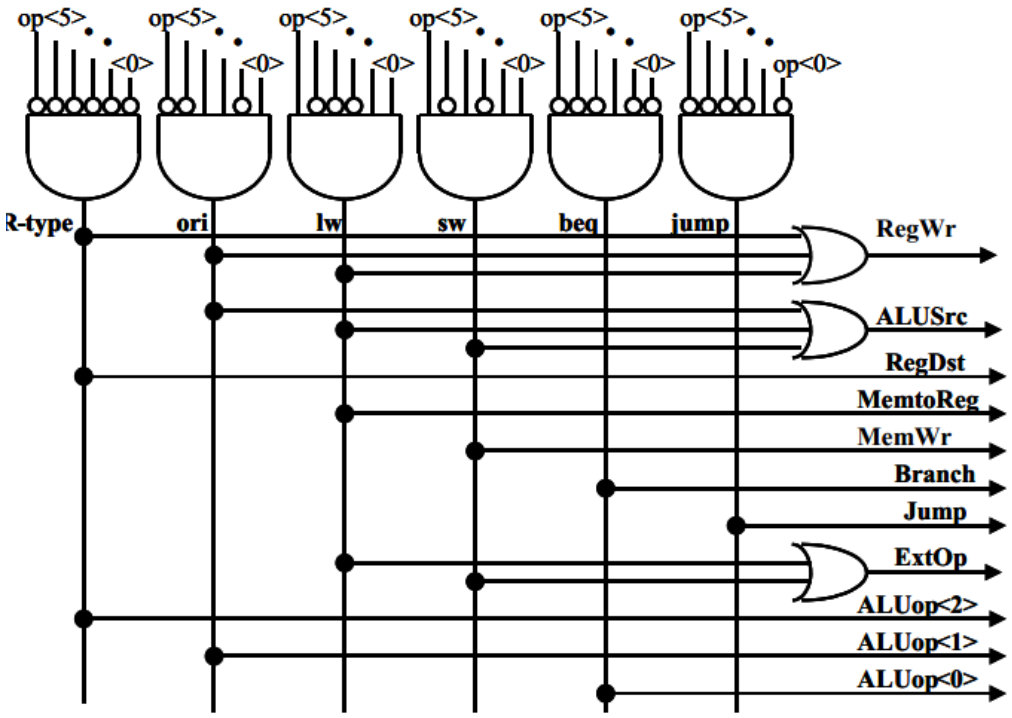
The Truth Table for ALUctr

ALUop (Symbolic)	R-type	ori	lw	sw	beq
	"R-type"	Or	Add	Add	Subtract
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01

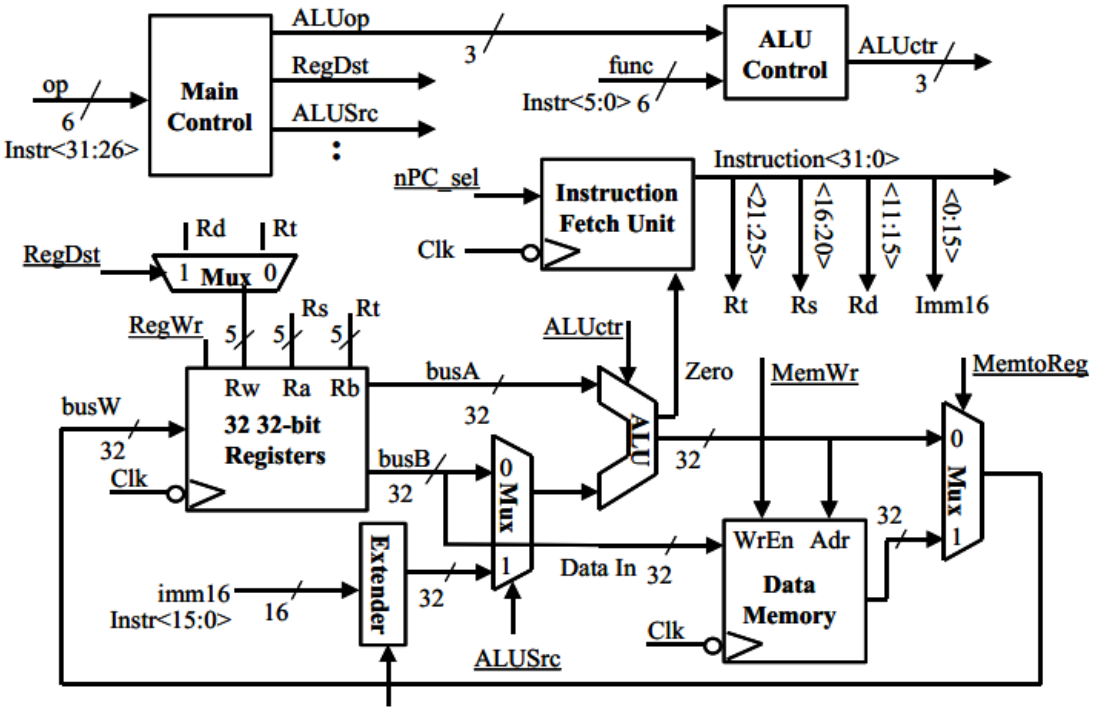
funct<3:0>	Instruction Op.
0000	add
0010	subtract
0100	and
0101	or
1010	set-on-less-than

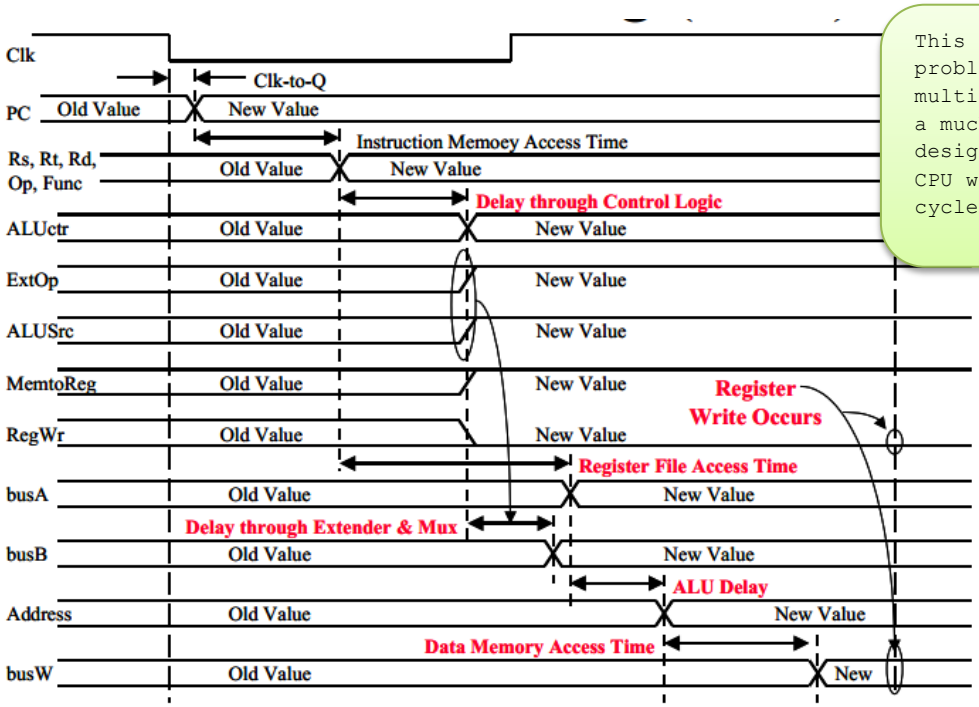
ALUop			func				ALU Operation	ALUctr		
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>		bit<2>	bit<1>	bit<0>
0	0	0	x	x	x	x	Add	0	1	0
0	x	1	x	x	x	x	Subtract	1	1	0
0	1	x	x	x	x	x	Or	0	0	1
1	x	x	0	0	0	0	Add	0	1	0
1	x	x	0	0	1	0	Subtract	1	1	0
1	x	x	0	1	0	0	And	0	0	0
1	x	x	0	1	0	1	Or	0	0	1
1	x	x	1	0	1	0	Set on <	1	1	1

PLA Implementation of Control Signals



The Single Cycle Processor and Worst Case Timing

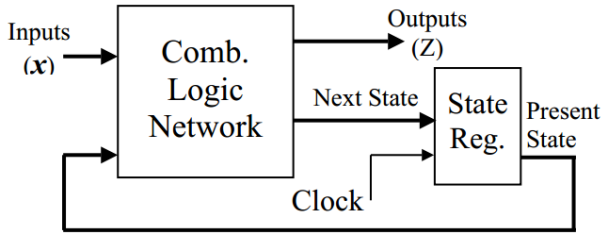




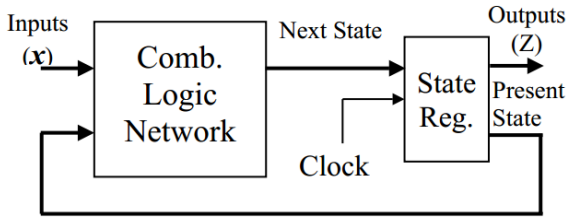
This CPU has many problems and the multi cycle CPU is a much better design. The lab 4 CPU was single cycle.

Sequential Circuit Types

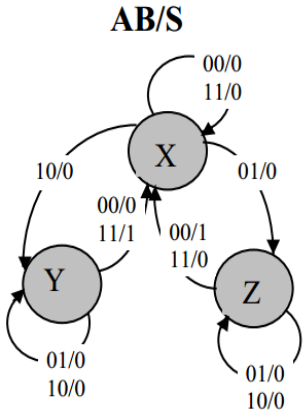
Mealy Machines: Their outputs depend on both the present state and the present inputs.



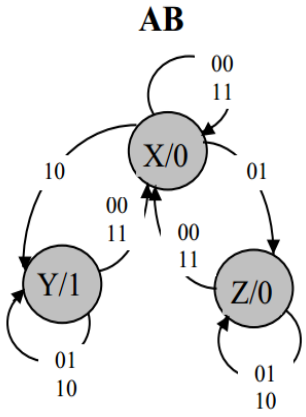
Moore Machines: The outputs depend on the present state only.



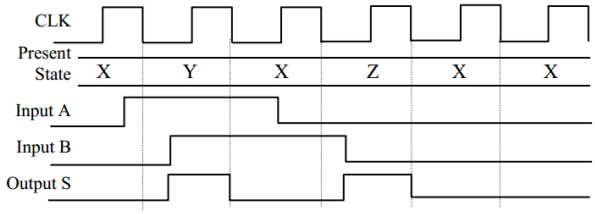
Mealy Model



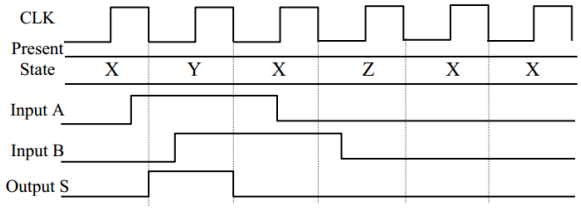
Moore Model



Mealy Model Timing Diagram



Moore Model Timing Diagram



PS	NS				Output
	AB				S
	00	01	11	10	
x	x/0	z/0	x/0	y/0	0
y	x/0	y/0	x/1	y/0	1
z	x/1	z/0	x/0	z/0	0

Mealy State Table

PS	NS				Output
	AB				S
	00	01	11	10	
x	x	z	x	y	0
y	x	y	x	y	1
z	x	z	x	z	0

Moore State Table

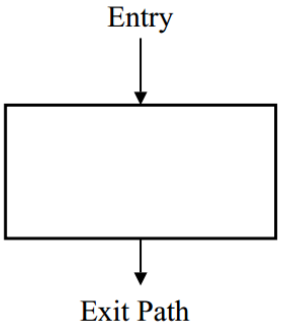
To realize mealy state diagrams, Kmaps are used to simplify the outputs and then the final equation is used to realize the output.

Algorithmic State Machine

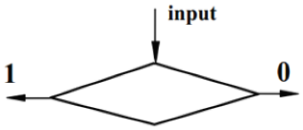
Flow charts can be used to create ASMs and implement them into hardware.

ASM Chart Model

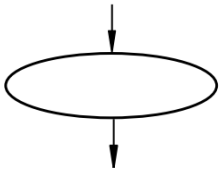
State Box



Decision Box



Conditional Output Box

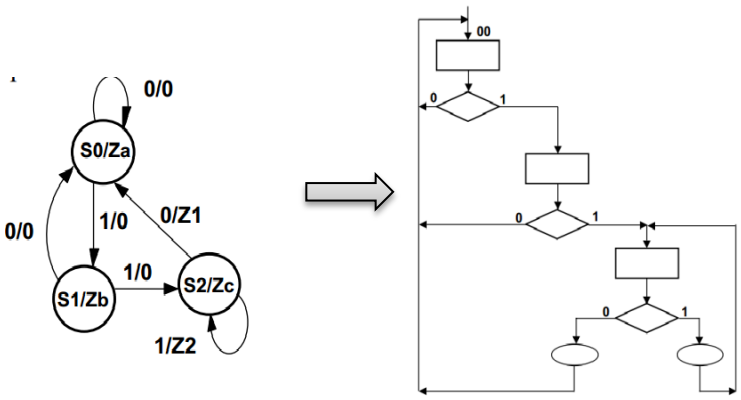


Rules to Construct an ASM Block

- ✓ For every valid combination of input variables, there must be one exit path.
- ✓ No internal feedback within an ASM block is allowed.
- ✓ An ASM block can have several parallel paths that lead to the same exit path and more than one these paths can be active at the same time.

Example of a chart to block model

- X is an input
- Za, Zb, Zc are Morore Outputs
- Z1 & Z2 are Mealy Outputs



ASM Chart to Equation

Working Example

For Next State: Consider Variable B Link-paths for States that has B = 1 are S1 and S2 states.

Link-Path-1

- Starting with a present state AB = 00, takes the X=1 branch and terminates at state S1 during which B = 1.

Link-Path-2

- Starting state 01, takes X=1 branch & ends at state 11.

Link-Path-3

- Starting at state 11, takes X=1 branch and ends in state 11.

Overall $B^+ = A'B'X + A'BX + ABX$

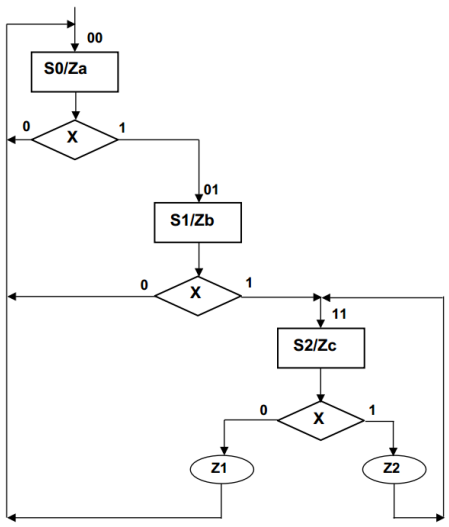
For Next State: Consider State Variable A.

- Two link paths terminate at S2 state

Moore Outputs

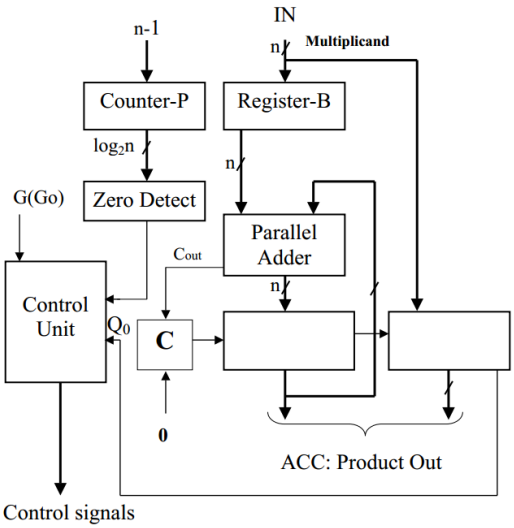
Za = A'B'; Zb = A'B; Zc = AB

Example

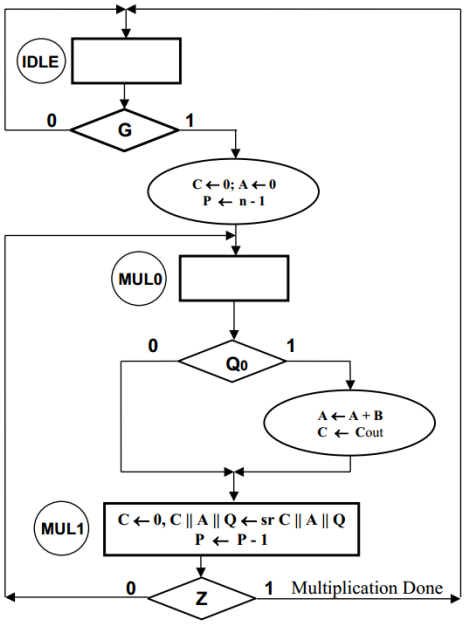


Multiplier

Block Diagram



Multiplier Control, ASM



sr = shift right and
C || A || Q ← sr C || A || Q is equivalent to 4 transfers
A(n-1) ← C, A ← sr A, Q(n-1) ← A(0), Q ← sr Q

Multiplier Control Unit

Control Unit is the Heart of Multiplier

- Its input, G initiates multiplication.
- It uses Q_0 (LSB of Q shift register) and counter zero-detect, Z signals.
- Control unit generates control signals to activate following micro-operations:
  - Sum of A and B.
  - PP transferred to A.
  - C_{out} transferred to C.
  - PP & multiplier in A:Q shifted right.
  - Carry from C is shifted to MSB of A:
    - LSB of Q is discarded.
    - After right shift, 1-bit of PP is transferred into Q and multiplier bits are shifted one bit right.
  - Control unit decides between add-shift and shift depending on the LSB of Q.
  - Control unit checks Z for an end.
  - Control unit checks G, to start multiplication.

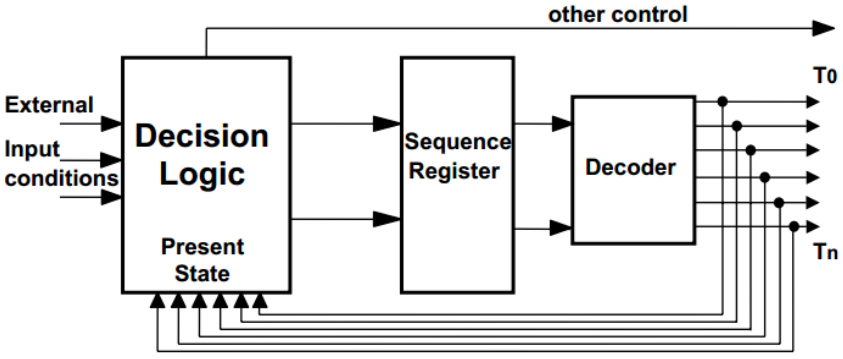
Control Signals for Multiplier

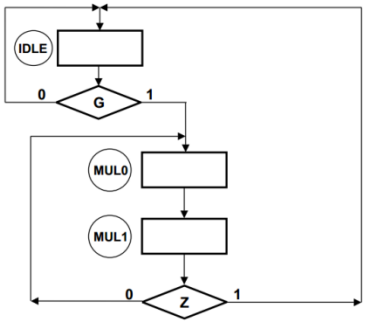
Micro-operations for each register

Block Diag Module	Micro-Operation	Control Signal	Control Expression
Register A	A ← 0 A ← A + B C‖A‖Q←sr C‖A‖Q	Initialize Load Shift	
Register B	B ← IN	Load_B	
FF C	C ← 0 C ← Cout	Clear_C Load	
Register Q	Q ← IN C‖A‖Q←sr C‖A‖Q	Load_Q Shift	
Counter P	P ← n – 1 P ← P – 1	Initialize Decrement Count	

Sequence Register and Decoder Method

Decoder provides outputs corresponding to each state. Register with n FF's can have 2^n states, n-bit sequence register has n-FF's and associated gates.

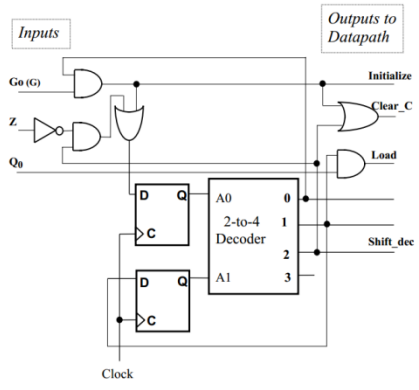




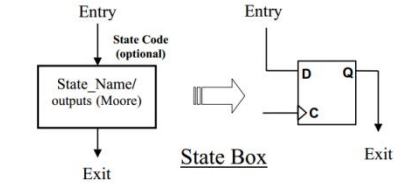
State Table

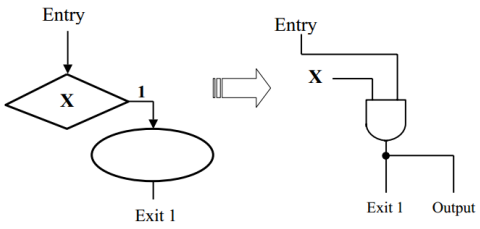
Present State		Inputs		N. State		Decoder		
Name	M1 M0	G	Z	M1+ M0+	IDLE	MUL0	MUL1
IDLE	0 0	0	x		1	0	0
	0 0	1	x		1	0	0
MUL0	0 1	x	x		0	1	0
MUL1	1 0	x	0		0	0	1
	1 0	x	1		0	0	1
	1 1	x	x		x	x	x

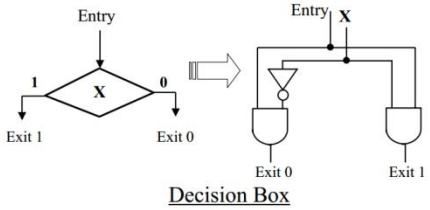
Implementation

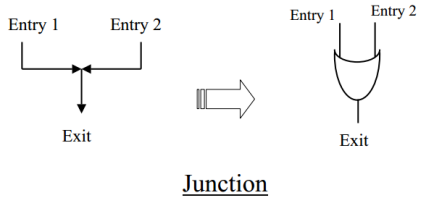


ASM Transforming Rules

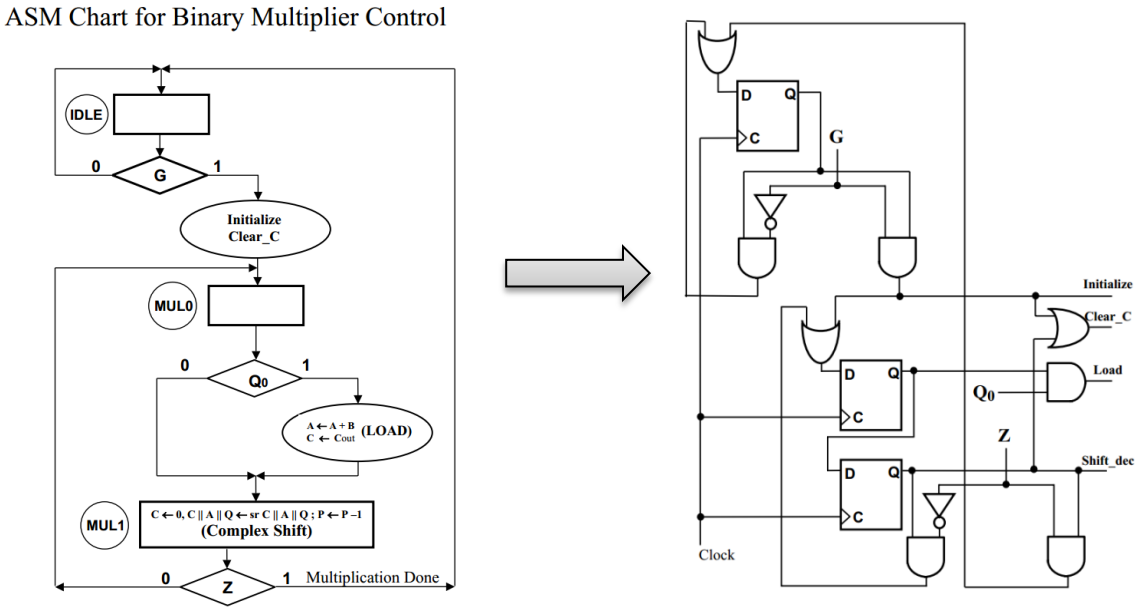








Example (Using the rules above)



MIPS-Lite Multi-cycle Control

The Multi-cycle Approach:

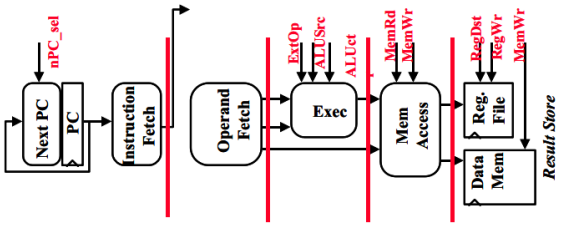
- The ALU is used to compute address and increment the PC
- Memory used for instruction and data
- Control signals will not be determined solely by instructions
- Control unit design by using classical FSM design is impractical due to large number of inputs and states
- An extension to the classical approach is used by experienced designer in designing control logic circuits
    - Sequence reg and decoder method
    - One FF per state method
    - Micro-program controller
    - PLA controller (uses PLD ROM/PROM)

The main idea of the multi-cycle is to change the data-path into segments so some instruction can run faster, so it can support pipelining and so it can support a new control unit design. Essentially the instructions will be broken into steps.

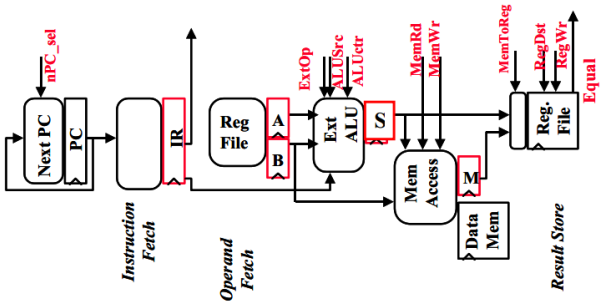
Partitioning Single-Cycle Datapath into parts

The steps for the multicycle Datapath are:

1. I-fetch
2. Reg Fetch / I-decode
3. Execute
4. Memory Read
5. Write

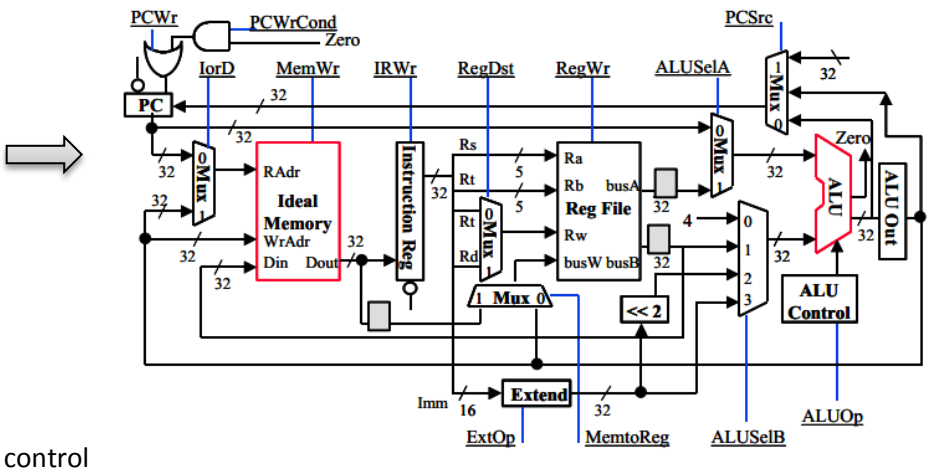


Multicycle Datapath



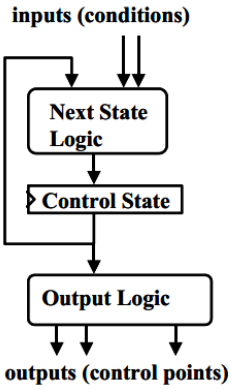
Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR = Memory[PC] PC = PC + 4			
Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A ==B) then PC = ALUOut	PC = PC [31-28] II (IR[25-0]<<2)
Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

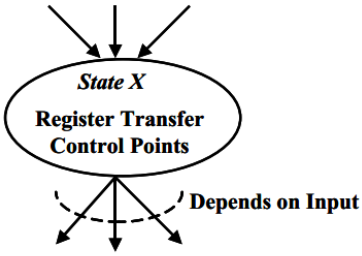
The Final Product

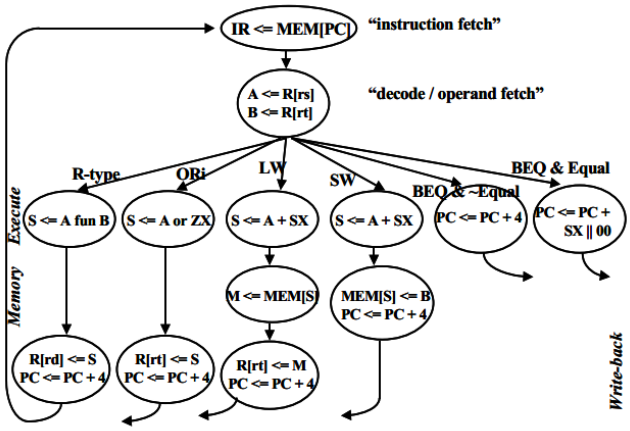


Control Model

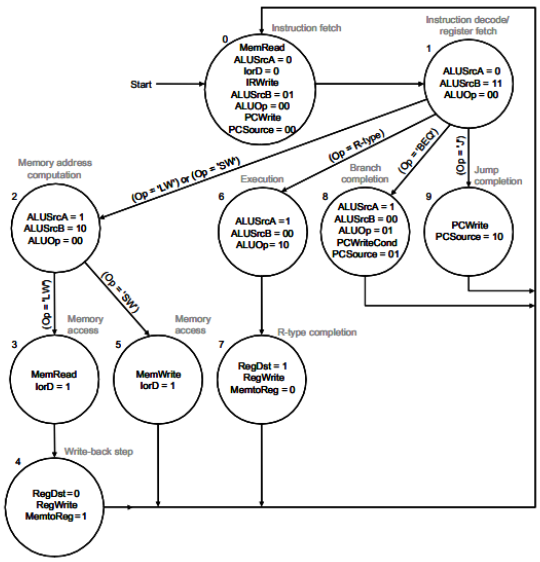
The state specifies control points for register transfer. Transfers occur upon exiting the state (falling edge) i.e control outputs are of Mealy type.







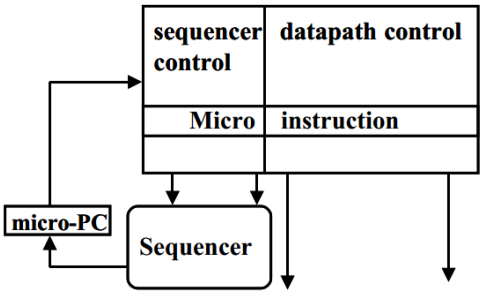
FSM for Datapath Control



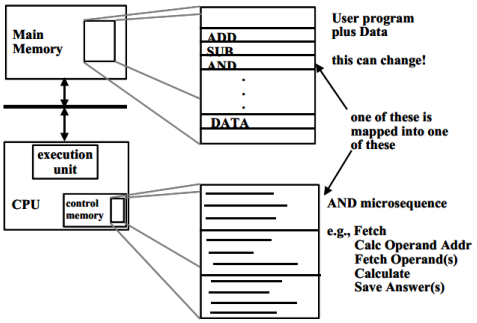
Microprogram Control Design

The state diagrams define the controller for an instruction set processor are highly structured.

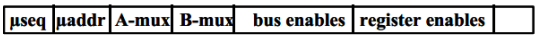
- Use this structure to construct a simple "micro-sequencer"
- Control reduces to programming this very simple device using the concept of: microprogramming



Processor Instruction Interpretation



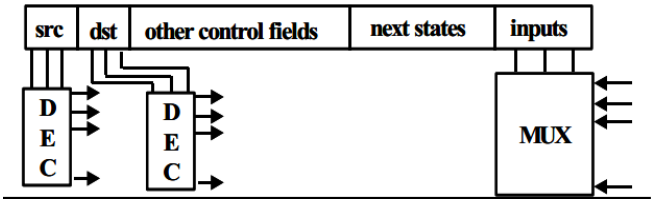
"Horizontal" Microcode



- Control field for each control point in the machine.
- Lots of control states but provide more control over the parallelism in the datapath.

"Vertical" Microcode

- Compact microinstruction format for each class of µ-operation.
- Local decode to generate all control points.
- Extra level of decoding can slow down the machine.

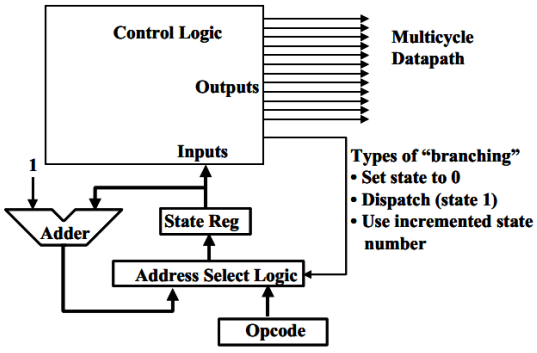


Microprogram-based Control

Control is the hard part of processor design:

- Datapath is fairly regular and well organized
- Memory is highly regular
- Control is irregular and global

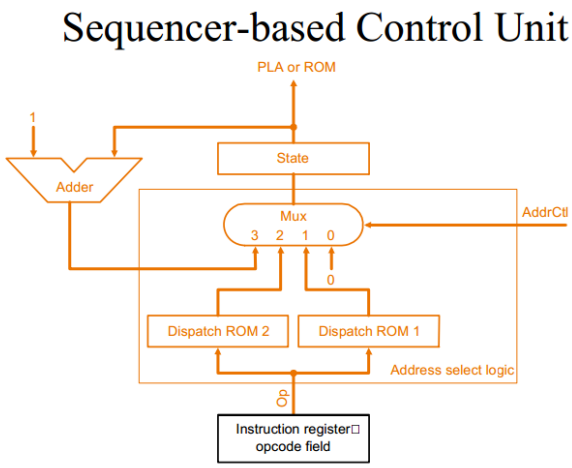
Sequencer-based Control Unit



Microinstruction Set Design

- Start with a list of control signals
- Group signals together that make sense (this is called fields)
- Place fields in some logical order (example: ALU operations and ALU operations first)
- Create a symbolic legend for u-instruction format
- Use computers to design computers. U-assembler to get binary code
- To minimize the width, encode ops that will never be used at the same time

Field name	Value	Signals active	Comment
ALU control	Add	ALUOp = 00	Cause the ALU to add.
	Subt	ALUOp = 01	Cause the ALU to subtract; this implements the compare for branches.
	Func code	ALUOp = 10	Use the instruction's function code to determine ALU control.
SRC1	PC	ALUSrcA = 0	Use the PC as the first ALU input.
	A	ALUSrcA = 1	Register A is the first ALU input.
SRC2	B	ALUSrcB = 00	Register B is the second ALU input.
	4	ALUSrcB = 01	Use 4 as the second ALU input.
	Extend	ALUSrcB = 10	Use output of the sign extension unit as the second ALU input.
	Extshift	ALUSrcB = 11	Use the output of the shift-by-two unit as the second ALU input.
Register control	Read		Read two registers using the rs and rt fields of the IR as the register numbers and putting the data into registers A and B.
	Write ALU	RegWrite, RegDst = 1, MemtoReg = 0	Write a register using the rd field of the IR as the register number and the contents of the ALUOut as the data.
	Write MDR	RegWrite, RegDst = 0, MemtoReg = 1	Write a register using the rt field of the IR as the register number and the contents of the MDR as the data.
Memory	Read PC	MemRead, IorD = 0	Read memory using the PC as address; write result into IR (and the MDR).
	Read ALU	MemRead, IorD = 1	Read memory using the ALUOut as address; write result into MDR.
	Write ALU	MemWrite, IorD = 1	Write memory using the ALUOut as address, contents of B as the data.
PC write control	ALU	PCSource = 00 PCWrite	Write the output of the ALU into the PC.
	ALUOut-cond	PCSource = 01, PCWriteCond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	jump address	PCSource = 10, PCWrite	Write the PC with the jump address from the instruction.
Sequencing	Seq	AddrCtl = 11	Choose the next microinstruction sequentially.
	Fetch	AddrCtl = 00	Go to the first microinstruction to begin a new instruction.
	Dispatch 1	AddrCtl = 01	Dispatch using the ROM 1.
	Dispatch 2	AddrCtl = 10	Dispatch using the ROM 2.

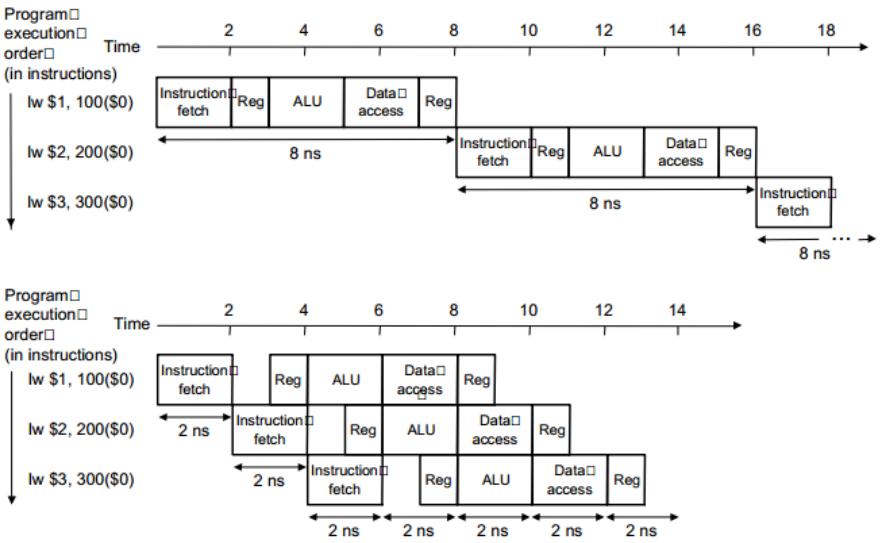


Example for Fetch and Decode

Label	ALU Control	SRC1	SRC2	Reg. Control	Mem	PCWrite Control	Sequence
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Ext shift	Read			Dispatch-1

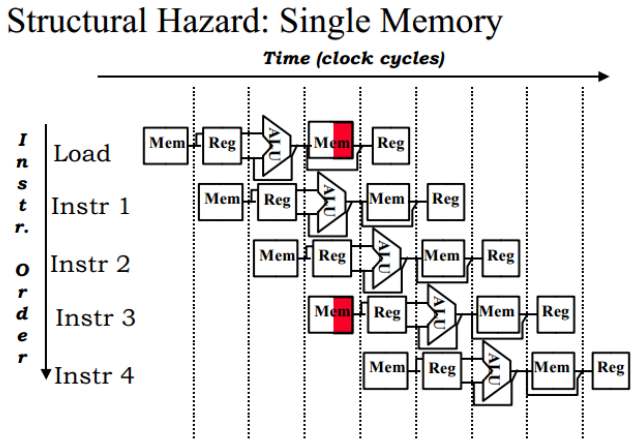
Pipelining

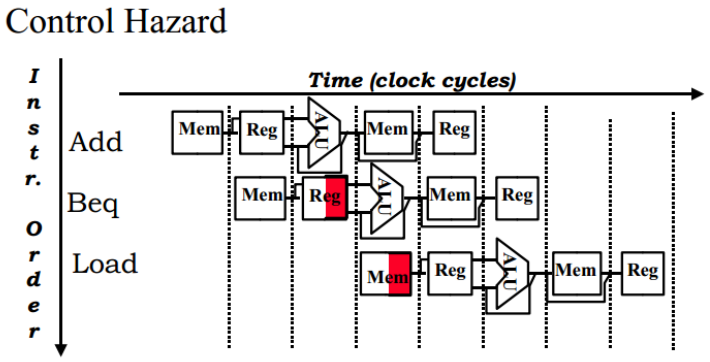
Pipelining is simply taking each step and when the step has completed, the next instruction can use the step and create an assembly line structure in the CPU. There are many hazards to pipelining so most of the design involved hazard control.



Hazards

Structural Hazards: attempt to use the same resource in two different ways at the same time
Data Hazards: attempt to use an item before it is ready (instruction depends on result of prior)
Control Hazards: attempt to make a decision before condition is evaluated (branches)

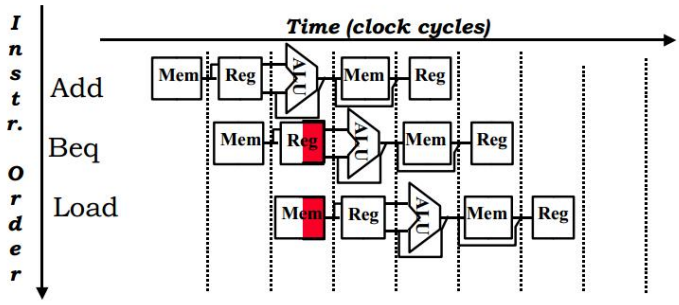




Control Hazard Solutions

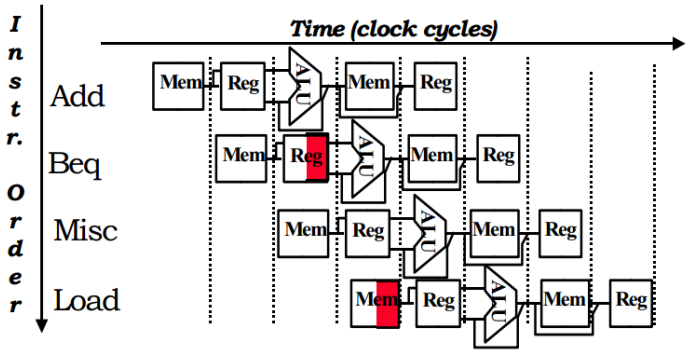
Stall

It is possible to move the decision upward to 2nd stage by adding hardware that checks the registers as being read.



- Predict: guess one branch direction then backup if wrong.

Redefine branch behavior (takes place after next instruction) "delayed branch"

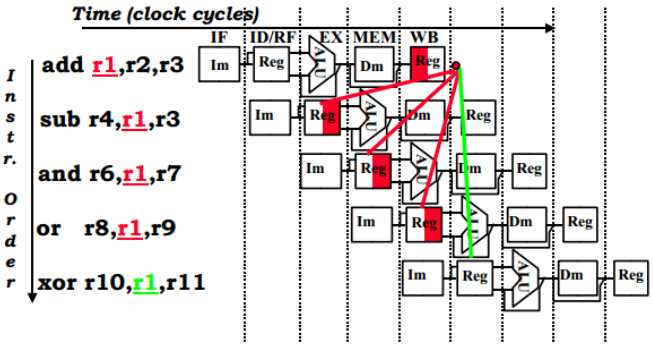


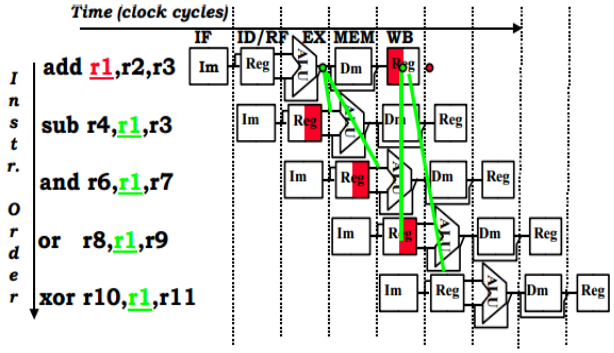
- Impact:
  No clock cycle is wasted if one can find an instruction to put in "slot"

Data Hazard and Solution

Data Hazard in r1

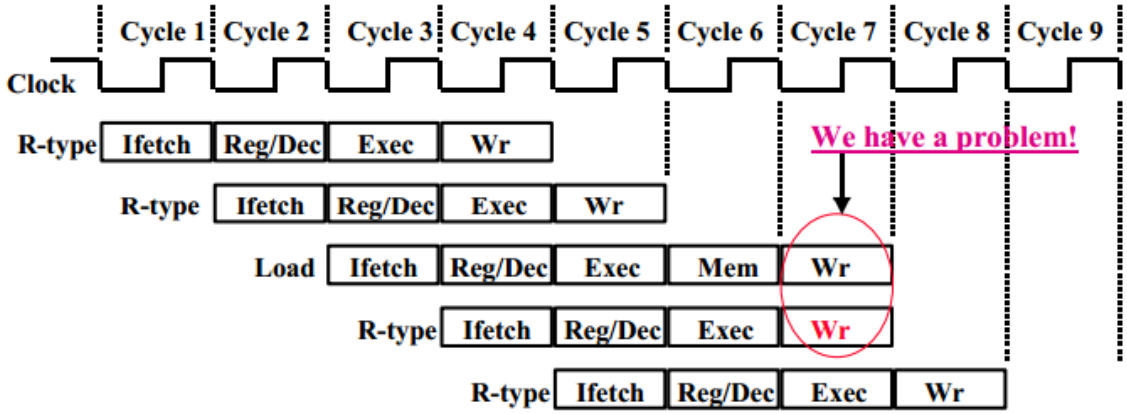
Dependencies backwards in time are hazards



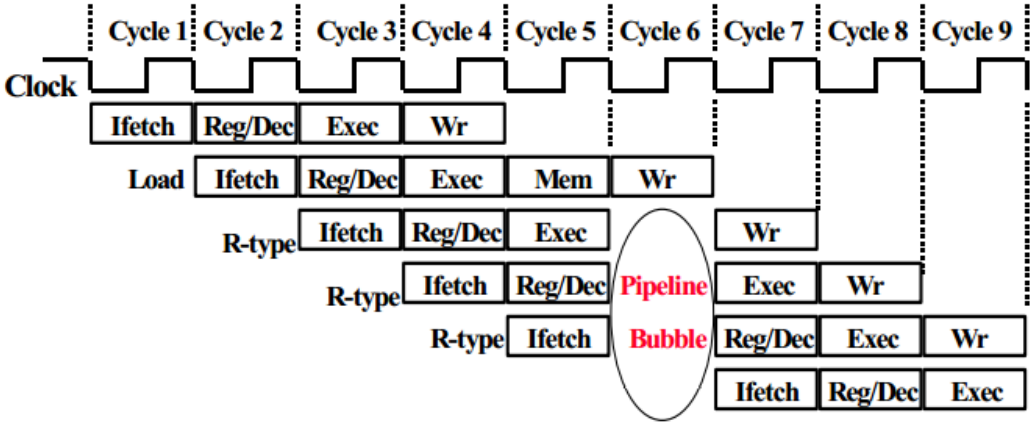


"Forward" result from one stage to another.

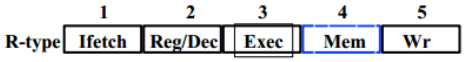
Structural Hazard and Solution

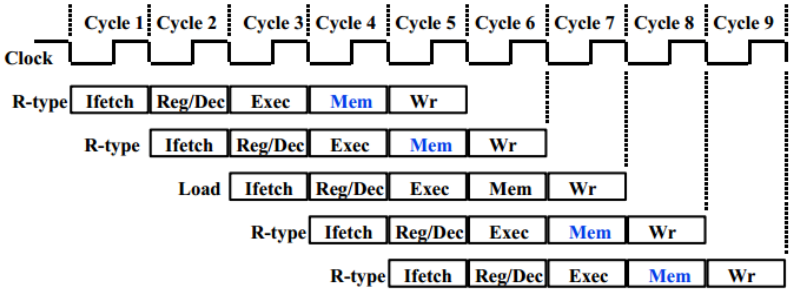


To fix this we can insert a "Bubble" into the pipeline which essentially is a delay. This has complex control logic.



Also, since R-type instructions have no Mem stage due to them being register only instructions, a memory (NOOP) can be added where nothing happens but this will prevent many structural hazards.

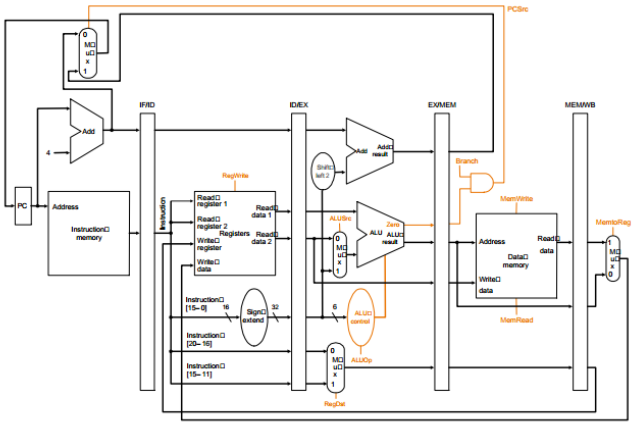




How do we split the Datapath into stages?

Pipeline Control

Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X



Main Control generates the control signals during Instruction Decode

- Exec (ExtOp, ALUSrc, ...) control signals are used 1 cycle later.
- Mem (MemWr Branch) control signals are used 2 cycles later.
- Wr (MemtoReg MemWr) control signals are used 3 cycles later.

Pass control signals along just like the data