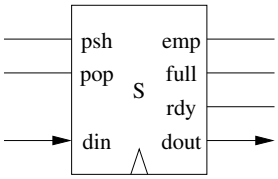
10 pages CMPT 250: Sample Final Exam SOLN SAMPLE

1. **VHDL Programming:**

(a) The following behavioral description defines a type of memory called a "stack memory" or "push-down store". From this definition, provide a VHDL entity definition for the stack memory. Assume that the stack memory will provide 2^8 locations each capable of storing binary sequences of length 16: **(4 marks)**



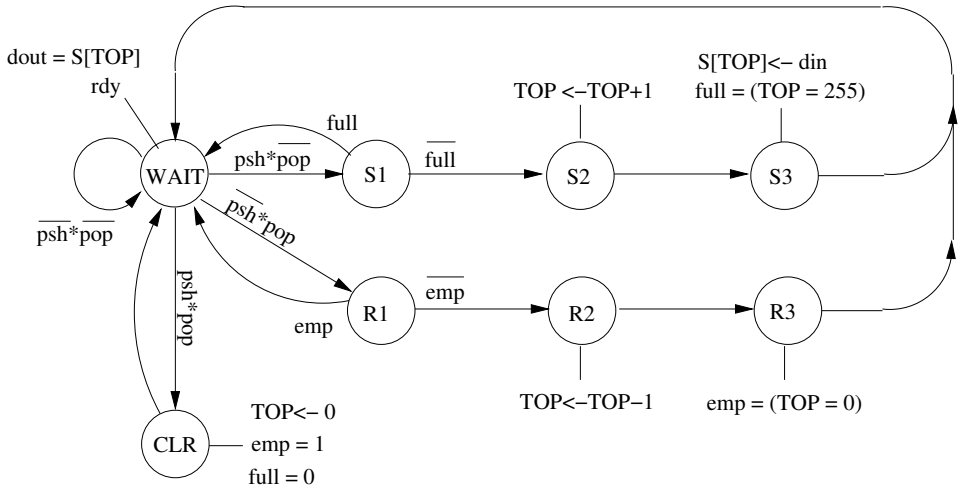
psh	pop	FUNCTION
0	0	none
0	1	TOP <– TOP – 1
1	0	TOP<–TOP+1, S[TOP]<– din
1	1	TOP<– 0

NOTE: In all cases, dout = S[TOP], emp = (TOP = 0), full = (TOP = 255)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity S is
   port( clk, push, pop : in std_logic;
         din : in std_logic_vector(15 downto 0);
         emp, full, rdy : out std_logic;
         dout : out std_logic_vector(15 downto 0));
end S;
```

(b) An algorithm for implementing the $2^8 \times 16$ stack memory is provided by the following state machine diagram:



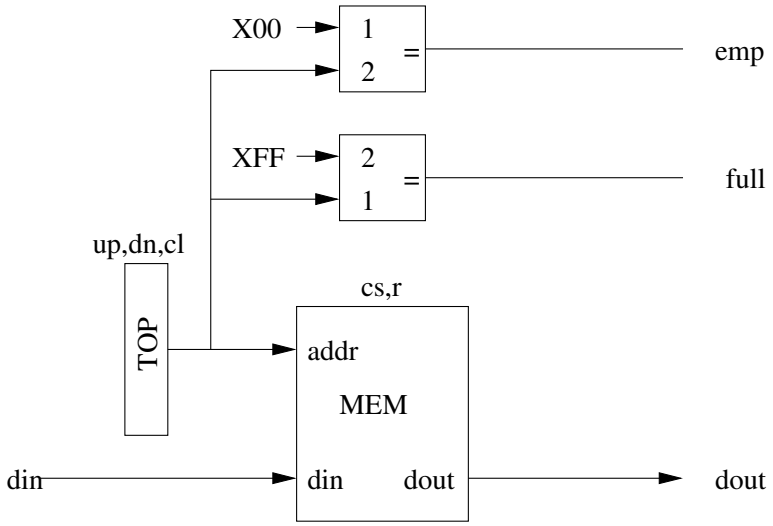
i. On the next two pages, implement the algorithm as a VHDL architecture to accompany your entity definition for the previous question. Your architecture body should include two processes: one to implement the state transitions and one to implement the asserted outputs. **(12 marks)**

```
architecture behav of S is
type state is (WT, S1, S2, S3, R1, R2, R3, CLR);
type stack_memory is array (0 to 255) of std_logic_vector(15 downto 0);
signal next_state : state;
signal TOPsig : natural;                        -- OPTIONAL

begin
-- State Transition process:
process is
variable curr_state :state := WT;
begin
   if clk = '1' then
      case curr_state is
         when WT =>
            if (push and pop) = '1' then curr_state := CLR;
            elsif (push and not pop) = '1' then curr_state := S1;
            elsif (not push and pop) = '1' then curr_state := R1;
            end if;
         when S1 =>
            if full = '0' then curr_state := S2;
            else curr_state := WT;
            end if;
         when S2 =>
            curr_state := S3;
         when S3 | R3 | CLR =>
            curr_state := WT;
         when R1 =>
            if emp = '0' then curr_state := R2;
            else curr_state := WT;
            end if;
         when R2 =>
            curr_state := R3;
      end case;
      next_state <= curr_state;
   end if;
   wait on clk;
end process;
```

```
-- Asserted Outputs process:
process is
variable TOP : natural := 0;
variable S: stack_memory;
variable rdy_val : std_logic;
begin
   rdy_val := '0';
   case next_state is
      when WT =>
         dout <= S(TOP);  rdy_val := '1';
      when S1 =>
      when S2 =>
         TOP := (TOP + 1) mod 256;            -- mod 256 optional
      when S3 =>
         S(TOP) := din;
         if TOP >= 255 then full <= '1';
         end if;
         emp <= '0';
      when R1 =>
      when R2 =>
         TOP := (TOP - 1) mod 256;            --mod 256 optional
      when R3 =>
         if TOP = 0 then emp <= '1';
         end if;
         full <= '0';
      when CLR =>
         TOP := 0;
         emp <= '1';
         full <= '0';
   end case;
   rdy <= rdy_val;
   TOPsig <= TOP;                             -- OPTIONAL
   wait on next_state;
end process;
end behav;
```

ii. Draw the diagram of a datapath derived from the state machine diagram using a "generic memory module" (`cs, r, din, dout`, and `addr` ports), an "up/down counter with clear" (`up, dn`, and `clr` control inputs), and comparators. Your diagram should show the sizes of any buses and all external inputs and outputs should be labelled. **(4 marks)**



iii. Complete the following function table to define the control point enabler for the stack memory: **(8 marks)**

STATE		cs	r	up	dn	clr	rdy
WAIT	000	1	1	0	0	0	1
S1	001	0	0	0	0	0	0
S2	010	0	0	1	0	0	0
S3	011	1	0	0	0	0	0
R1	100	0	0	0	0	0	0
R2	101	0	0	0	1	0	0
R3	110	0	0	0	0	0	0
CLR	111	0	0	0	0	1	0

2. **CPU Design**:
Stack memories can be used to provide work storage in the design of CPU's instead of the usual register file. Consider the following instruction set:

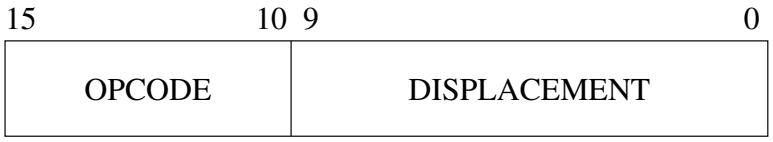
OPCODE	INST	OPND	FUNCTION	address mode	fetch accesses	execute accesses
000000	LD	opnd	AC ← M[opnd + PC]	index	1	1
000001	STO	opnd	M[opnd + PC] ← AC	index	1	1
000010	ADD		AC ← AC + S[TOP]	implied	1	0
000011	AND		AC ← AC *and* S[TOP]	implied	1	0
000100	CMPL		AC ← *not* AC	implied	1	0
000101	SUB		AC ← AC − S[TOP]	implied	1	0
000110	PSH		S[TOP] ← AC	implied	1	0
000111	POP		AC ← S[TOP]	implied	1	0
001000	BEQ	opnd	if AC = S[TOP] then PC ← opnd + PC	relative	1	0
001001	BEMP	opnd	if emp = 1 then PC ← opnd + PC	relative	1	0

"S" denotes a stack memory and the top element in the stack, S[TOP], is used to provide the second operand in the arithmetic/logic instructions. As well, instructions are provided to push data onto the stack memory (PSH) and to remove data from the stack memory (POP). In each case the register "AC" is used as either the source of the value pushed onto the stack memory, or the destination of the value popped from the stack memory.

(a) Propose a single instruction format suitable for accommodating all the instructions, assuming the following constraints:

- There will be a total of 48 instructions in the instruction set.
- The CPU will be interfaced with a $2^{16} \times 16$ memory.

Your format should show the location, size and purpose of each field within the instruction format. **(2 marks)**



(b) Based on your instruction format enter the following information beside the instructions in the table above:

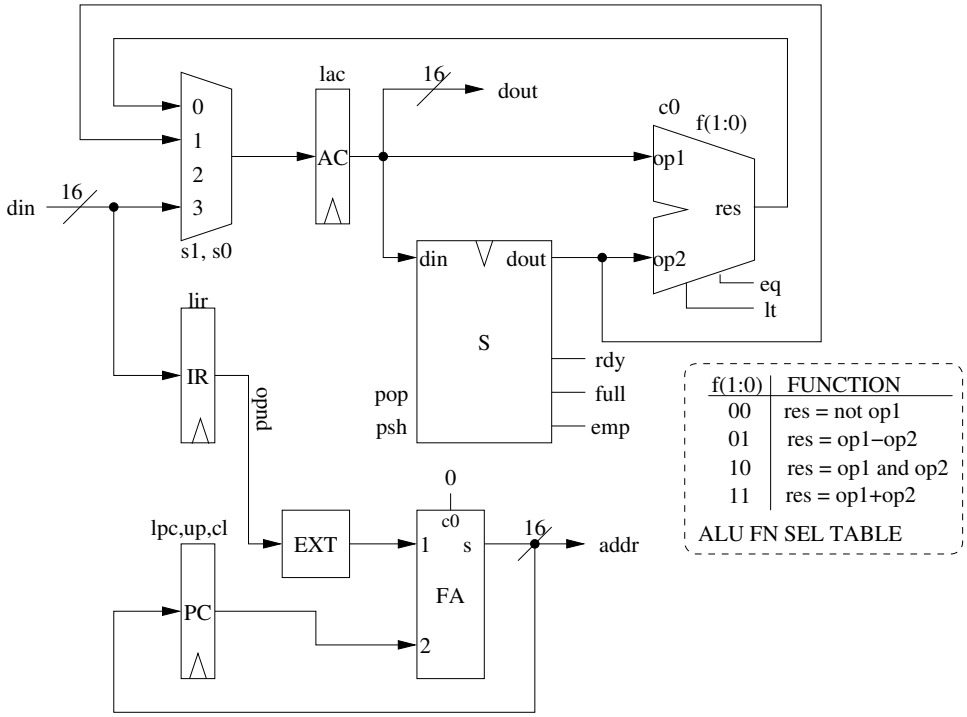
i. A suitable opcode, expressed in binary (base-2). **(1 mark)**

ii. The address mode of each instruction. **(3 marks)**

iii. The number of memory accesses that will occur when the instruction is retrieved. **(2 marks)**

iv. The number of memory accesses that will occur when the instruction is executed (after decoding). **(2 marks)**

(c) A datapath that implements the instruction set given is as follows:

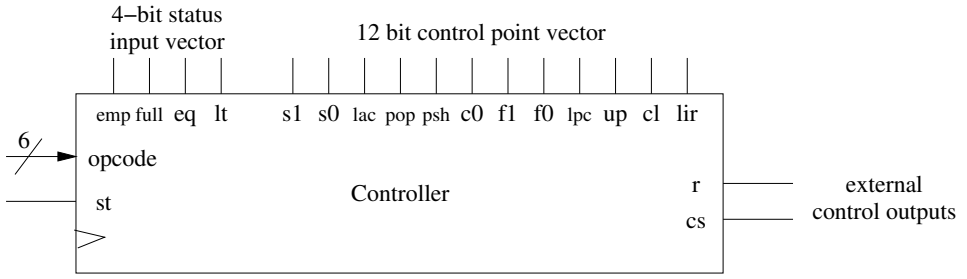


i. On the diagram next to each component label all the control inputs you would expect to find for that component (if any). **(4 marks)**

ii. For each of the following instructions identify all the control inputs that need to be asserted (i.e., set to 1) for the instruction to be executed correctly: **(4 marks)**

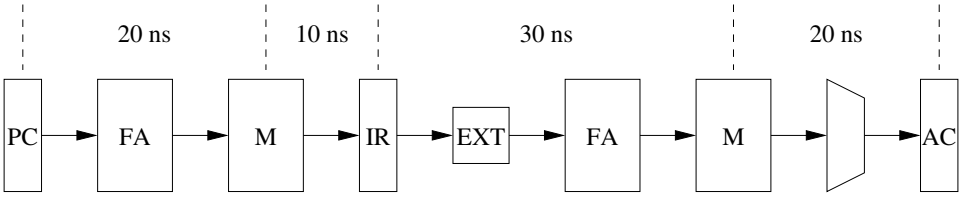
INSTRUCTION	ASSERTED CONTROL INPUTS
ADD	lac, f(1), f(0)
POP	pop, lac, s0
BEMP (emp=1)	lpc
LD 0	cs, r, s1, s0, lac

iii. Draw an entity diagram for the controller that would accompany this datapath. Label all ports. You may assume there is only one external control input, "`st`" which is used to initiate execution of the CPU and that the $2^{16} \times 16$ memory to which the CPU is connected has a standard generic interface. **(4 marks)**

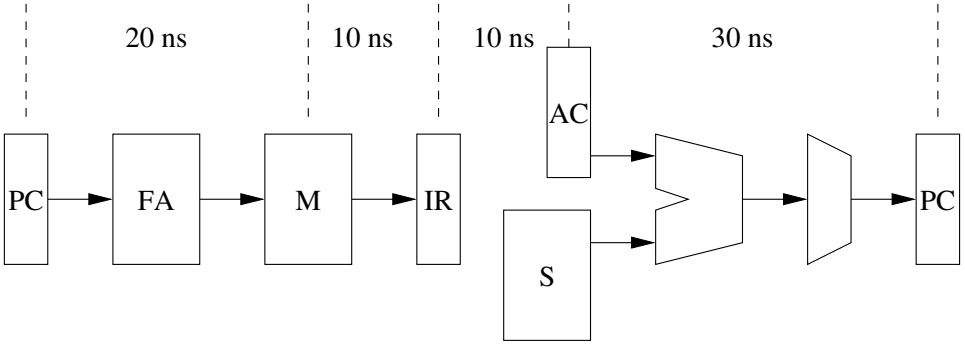


iv. Draw unfolded dataflow diagrams (UDF's) for the following instructions. In each diagram label the components and partition each UDF diagram into stages.

A. **LD 0**: **(3 marks)**



B. **SUB**: **(3 marks)**



v. Based on just these two instructions, what is the minimum clock period, if all components have the same propagation delay of 10 ns and what will be the execution time of each instruction? **(1 mark)**

ANSWER: 30 ns clock period. Execution time is 4 stages $\times$ 30 ns = 120 ns

vi. You have two registers. After which components of the original datapath would you place them in order to improve the execution time of each instruction and what would the revised execution time be? **(2 marks)**

ANSWER: After the ALU and after the EXT components. Clock period is now 20 ns and with 5 stages, execution times would be 100 ns.

3. **Memory:**

(a) A DRAM has a memory cycle time of 100 ns, which includes a refresh of one row of its storage matrix. If the refresh period of its storage elements is 1 millisecond, what is the maximum size of the DRAM, assuming there are 2^{10} bits in each row? You may use the approximation "$2^{10} \approx 10^3$" in your answer. **(2 marks)**

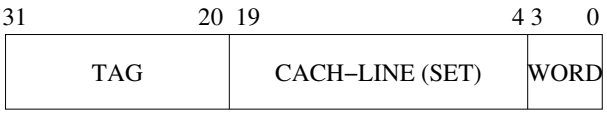
ANSWER:

$$\begin{aligned} 1024 \times \frac{10^6}{10^2} &= 1024 \times 10^4 \\ &\approx 10^3 \times 10^4 \\ &\approx 10^7 \\ &\approx 10 \times 2^{20} \end{aligned}$$

(b) A $2^{32} \times 16$ cache memory employing a direct mapping strategy with write-back update is to be designed. A $2^{16} \times n$ SRAM is to be used to provide the cache component, where n is to be determined. If main memory is implemented with DRAM employing a blocking factor of 16, provide the address mapping format and cache line format for the memory. Indicate the position, size and name of each field in your formats. **(6 marks)**

ANSWER:
Number of blocks of main memory: $2^{32}/2^4 = 2^{28}$
Number of blocks per cache line set : $2^{28}/2^{16} = 2^{12}$
Formats:

Address Mapping Format



Cache Line Format

(c) During the execution of a program, the following blocks of memory were accessed in the order indicated:

08, 09, 0A, 0B, 08, 0A, 0C, 08, 0E, 0A, 0E, 0C

Assuming a cache memory with four cache lines, complete the memory mapping trace tables for each of the following address mapping strategies:

i. Direct Mapping **(3 marks)**

ACCESS NUMBER

CACHE LINE	0	1	2	3	4	5	6	7	8	9	10	11	12
0	--	08	08	08	08	08	08	0C	08	08	08	08	0C
1	--	--	09	09	09	09	09	09	09	09	09	09	09
2	--	--	--	0A	0A	0A	0A	0A	0A	0E	0A	0E	0E
3	--	--	--	--	0B	0B	0B	0B	0B	0B	0B	0B	0B
						hit	hit						

ii. Fully Associative Mapping with LRU Replacement: **(3 marks)**

ACCESS NUMBER

CACHE LINE	0	1	2	3	4	5	6	7	8	9	10	11	12
0	--	(08)	08	08	08	(08)	08	08	(08)	08	08	08	08
1	--	--	(09)	09	09	09	09	(0C)	0C	0C	0C	0C	(0C)
2	--	--	--	(0A)	0A	0A	(0A)	0A	0A	0A	(0A)	0A	0A
3	--	--	--	--	(0B)	0B	0B	0B	0B	(0E)	0E	(0E)	0E
						hit	hit	hit			hit	hit	hit

(d) For each example what is the hit ratio? **(2 marks)**

ANSWER: For direct mapping, hit ratio = 2; for associative mapping, hit ratio = 6

4. **Virtual Memory:**
A virtual memory system is comprised of a $2^{32} \times 32$ DRAM main memory and a disk drive providing 2^{36} sectors of $1\text{K} = 2^{10}$ bytes.

(a) If a page of virtual memory consists of 16K locations of 32-bit words, how many sectors are required to store a page on the disk drive? **(2 marks)**

ANSWER:

$$\begin{aligned} 1 \text{ sector} &= 2^{10} \text{ bytes} \\ &= 2^8 \text{ words (4 bytes per word)} \end{aligned}$$

$$\begin{aligned} 1 \text{ page} &= 2^{16} \text{ words} \\ \text{no. of sectors/page} &= 2^{16}/2^8 \\ &= 2^8 \text{ sectors} \end{aligned}$$

(b) As a power of 2, what is the size of the virtual address space? That is, how many bits are required to specify a virtual address if a 32-bit word can be stored at that address? **(1 mark)**

ANSWER:

$$\begin{aligned} 2^{10} \text{ bytes/sector} \times 2^{36} \text{ sectors} &= 2^8 \text{ words/sector} \times 2^{36} \\ &= 2^{44} \text{ virtual addresses} \end{aligned}$$

Therefore, 44 bits are required to specify a virtual address.

(c) How many pages of virtual memory are there? **(1 mark)**

ANSWER: $2^{44}/2^{16} = 2^{28}$ pages

(d) How many pages are there in the page table of any process? **(1 mark)**

ANSWER : 2^{28} pointers requires $2^{28}/2^{16} = 2^{12}$ pages

(e) How many pages will be required for the directory table of each process? **(1 mark)**

ANSWER: $2^{12}/2^{16} = 1$ page.

(f) Provide an address mapping format for mapping a virtual address to a physical one. Indicate the size, position, and purpose of each field. **(3 marks)**

ANSWER: