COL 216 Computer Architecture : Major Test

Date: 08.04.2022 Time: 14:15-16:15 Max marks: 30

***Note**: Do any one out of questions 1 and 2. Do all of questions 3 to 5.*

1. Compare computation of statement s = (a + b) – (c + d) in three processors P1, P2 and P3 defined below in terms of number of instruction bytes and data bytes transferred between CPU and memory. Here a, b, c, d and s refer to memory locations. The expression is to be evaluated as it is, without any algebraic transformation or re-ordering of operations, that is, left addition followed by right addition followed by subtraction.

Proc	Processor type	Instruction size, data size	Relevant Instructions
P1	accumulator based, 1-address	32-bit, 32-bit	load x ACC ← Mem[x] store x Mem[x] ← ACC add x ACC ← ACC + Mem[x] sub x ACC ← ACC – Mem[x]
P2	M-M type, 3-address	64-bit, 32-bit	add x, y, z Mem[x] ← Mem[y] + Mem[z] sub x, y, z Mem[x] ← Mem[y] – Mem[z]
P3	R-R type, 3-address, with registers r0 ... r15	32-bit, 32-bit	load Rd, x Rd ← Mem[x] store Rd, x Mem[x] ← Rd add Rd, Rn, Rm Rd ← Rn + Rm sub Rd, Rn, Rm Rd ← Rn – Rm Rd, Rn, Rm ∈ {r0, r1, . . . , r15}

(8)

Solution:

	Processor P1	Processor P2	Processor P3

Program	load a
	add b
	store t1
	load c
	add d
	store t2
	load t1
	sub t2
	store s
	[2 marks]
		add t1, a, b
		add t2, c, d
		sub s, t1, t2
		[2 marks]
			load r1, a
			load r2, b
			add r3, r1, r2
			load r1, c
			load r2, d
			add r4, r1, r2
			sub r3, r3, r4
			store r3, s
			[1 mark]

Instruction bytes	9x4	6x4	8x4
Data bytes	9x4	9x4	5x4
Total bytes	18x4 [1 mark]	15x4 [1 mark]	13x4 [1 mark]

COL 216 Computer Architecture : Major Test

Date: 08.04.2022 Time: 14:15-16:15 Max marks: 30

2. Consider a multi-cycle processor design (like assignment 2, stage 3) executing ARM instructions {add, sub, cmp, mov, ldr, str, beq, bne, b} with limited variants. A new instruction icb (increment, compare and branch), as defined below, is to be added to this processor to facilitate implementation of loops.

   icb Rd, Rn, Rm Rd ← Rd + 1; if Rd < Rn then PC ← Rm else PC ← PC + 4

   What new control states would you introduce and what actions would be performed in those states? What changes/additions will be required in the glue logic? Whole design is not to be given - just write small fragments of VHDL code to make your answer about changes/additions precise. Make no change in register file (continue with 2 read and 1 write ports). Use ALU for all additions and comparisons. Assume that "1100" in bits 27-24 of instruction identify icb. Use any three 4-bit fields for specifying 3 registers.

   (8)

Solution:

This instruction will require at least 4 cycles - one instruction fetch cycle and at least three for reading Rd, incrementing Rd and writing back Rd. Other actions can be scheduled within these cycles, as shown below. Designs with larger number of states are acceptable.

1. fetch instruction
2. read Rd and Rn
3. increment Rd and read Rm
4. write back Rd and transfer Rm to PC if Rd less than Rn

To keep step 2 common with other instructions, let Rd and Rn be specified in bits 19-16 and 3-0 of the instruction, respectively. We can choose Rm to be specified in bits 15-12 of the instruction. Let the two control states that are common with other instructions be "fetch" and "read_AB". Two additional control states are "incr" and "wb_brn". Relevant state transitions of control FSM are defined below. these will appear in a clocked process. We assume that the instruction decoder defines a new instruction class "ICB" when instruction bite 27 downto 24 are "1100".

```
case control_state is
        when fetch =>    . . .  control_state <= read_AB;
        when read_AB =>
                . . .
                case instr_class is
                        when ICB => control_state <= incr;
                        . . .
                end case;
        when incr => . . .   control_state <= wb_brn;
        when wb_brn => . . .  control_state <= fetch;
        . . .
end case;
```

[2 marks for giving break up of actions into clock cycles/steps]
[1 mark for identifying new control states]
[1 mark for describing control state transitions]

COL 216 Computer Architecture : Major Test

Date: 08.04.2022 Time: 14:15-16:15 Max marks: 30

For defining actions, let the signals be named as follows.

Register file inputs and outputs:

RFread_addrA, RFread_addrB and RFwrite_addr are address inputs.
RFin, RFoutA and RFoutB are data input and outputs.
RFwrite_enable is write enable input.

ALU inputs and outputs:

operand1 and operand2 are operand inputs.
ALUout is the result output.
ALUop is the input specifying operation to be performed.
Carry_in is carry input.

MemoryOut is Memory output.

Program counter PC is considered as an internal signal rather than external component, along with IR, A, B and Res.

Relevant code fragments making assignments to IR, A, B, Res and PC and associated glue logic are given in the table below. Here type conversions are omitted.

Control state	Assignments to IR, A, B, Res and PC (inside the clocked process of control FSM)	Glue logic for inputs to ALU and RF (inside an unclocked process modeling a combinational circuit)
fetch => ALU performs PC(31..2) + 1	IR <= MemoryOut ; PC <= ALUout(29 downto 0)& "00";	ALUop <= adc; {add with carry} operand1 <="00"&PC_out(31 downto 2); operand2 <= X"00000000"; Carry_in <= '1';
read_AB => Rd, Rn read out from RF into A, B	A <= RFoutA; B <= RFoutB;	{RF addresses have default values}
incr => ALU performs A + 1 Rm read out from RF into A	Res <= ALUout; A <= RFoutA;	ALUop <= adc; {add with carry} {operand1 is A by default} operand2 <= X"00000000"; Carry_in <= '1'; RFread_addrA <= IR[15 downto 12];
wb_brn => ALU performs Res – B Res written into RF as Rd	if (ALUout[31] = '1') then PC <= A;	ALUop <= sub; {subtract} operand1 <= Res; {operand2 is B by default} RFwrite_addr <= IR[19 downto 16]; RFwrite_enable <= '1'; RFin <= Res;

Default inputs:

ALUop <= IR[24 downto 21]; operand1 <= A; operand2 <= B; Carry_in <= CarryFlag;

RFread_addrA <= IR[19 downto 16]; RFread_addrB <= IR[3 downto 0];
RFwrite_addr <= IR[15 downto 12]; RFwrite_enable <= '0'; RFin <= Res;

[2 marks for clocked assignments]
[2 marks for glue logic]

COL 216 Computer Architecture : Major Test

Date: 08.04.2022 Time: 14:15-16:15 Max marks: 30

3. Suppose we have a 5-stage pipelined processor with stages - IF (instruction fetch), ID (instruction decode/operand read), EX (execute), Mem (memory access) and WB (write back). What kind of hazards will be introduced if we have a single memory rather than separate program and data memories? How would you dynamically check for these hazards and insert stall cycles (ignore other types of hazards)? What would the number of stall cycles (average) depend on? Find an expression for average CPI accounting for the stall cycles.

(7)

Solution:

Type of hazard:
When instructions like load and store make an access to memory for reading/writing data, there will be resource conflicts with fetching of instructions. These are structural hazards or resource hazards.

[2 marks]

Checking for hazards and inserting stall cycles:
While fetching an instruction, check if the instruction in Mem stage is a load or store instruction. If yes, fetching of new instruction and incrementing of PC is prevented, thus inserting a stall cycle.

[2 marks]

Number of stall cycles:
Every load/store instruction will result in introduction of a stall cycle. Thus the total number of stall cycles will be equal to the number of load/store instructions executed. [Note: There is an exception to this - load/store instructions among the last three instructions executed will not cause this type of structural hazard. We will neglect this, assuming that the total number of instructions executed is much larger than 3.]

[1 mark]

CPI accounting for stall cycles:
Let N be the total number of instructions executed.
Let LS be the number of load/store instructions executed.
Number of cycles required to execute N instructions without stalls = N
(neglecting the last 4 cycles to drain the pipeline at the end, assuming that N is much larger than 4)
Number of stall cycles = LS
Total number of cycles = N + LS
Average CPI = (N + LS)/N or 1 + LS/N

[2 marks]

COL 216 Computer Architecture : Major Test

Date: 08.04.2022 Time: 14:15-16:15 Max marks: 30

4. (a) In a system with two levels of cache, what actions would need to be taken to ensure inclusiveness? What actions would need to be taken to ensure exclusiveness?

   (b) A fully associative TLB for a simple virtual memory has the following fields in each of its entries: valid bit, virtual page number and physical page number. How would you modify the TLB structure to support a virtual memory with two level page tables (that is, segmented virtual memory)? What would be various possible outcomes when this TLB is looked up?

   (7)

Solution:

(a) Two level inclusive/exclusive caches Let the two caches be referred to as L1 cache and L2 cache.

Inclusiveness implies that every block present in L1 is also present in L2. To ensure this, whenever a block in L2 is getting evicted, corresponding block in L1 must be flushed out or invalidated.

[1 mark]

Exclusiveness implies that the blocks that are present in L1 are not there in L2 and vice versa. To ensure this, the following actions are required. When there is L1 miss and L2 hit, the missing block is moved from L2 to L1 and its place in L2 is filled with the block displaced from L1. One can choose to discard the displaced block unless L1 is WB type and the block is dirty. If there is a miss at both levels, then the missing block is brought from memory and put in L1 only. The block displaced from L1 may be written back to L2.

[2 marks]

(b) TLB for segmented virtual memory

Here the virtual memory is divided into segments, which are divided into pages. At any time, the physical memory contains some pages of some segments. Each segment has a page table to keep track of pages of that segment. If a segment has no pages in physical memory, its page table also need not be there in the physical memory. A segment table keeps track of which page tables are in physical memory. From the segment table, we can find if and where the page table of the required segment is there in the physical memory. Then from the relevant page table, we can find if and where the required page is there in the physical memory.

Now the address is divided into three parts - segment number, virtual page number within the segment and offset within the page. For TLB to capture information about recently accessed pages, each entry should have a valid bit, segment number, virtual page number withing the segment and the corresponding physical page number. TLB should also capture information about recent accesses to the segment table. Each such entry should have a valid bit, segment number and the physical page number where the page table of that segment starts (assuming the page tables to be aligned at page boundaries). Combining these two, we can have the following structure of each TLB entry. The TLB could also be structured as two separate buffers to hold two types of entries.

- T: entry type (0 for segment plus page look-up, 1 for segment look-up)
- V: valid bit (0 for invalid, 1 for valid)
- seg: segment number
- vpn: virtual page number within the segment (for T = 0) or don't care (for T = 1)
- ppn: physical page number corresponding to segment seg and virtual page number vpn (for T = 0) or physical page number where the page table of segment seg starts (for T = 1)

[2 marks for taking care of both types of entries in one or two structures. 1 mark if only one type of entry is taken care of.]

COL 216 Computer Architecture : Major Test

Date: 08.04.2022 Time: 14:15-16:15 Max marks: 30

For a given virtual address <s, p, x> consisting of segment number s, virtual page number p and offset x, possible outcomes of TLB look-up are as follows.

(i) hit (segment plus page): TLB has an entry with T = 0, V = 1, seg = s, vpn = p and ppn = y, for some y.

(ii) hit (segment only): TLB does not have any entry of the kind (i) above, but has an entry with T = 1, V = 1, seg = s, vpn = q and ppn = y, for some q, y.

(iii) miss: TLB does not have any entry of the kind (i) or (ii) above.

[2 marks if both types of hits are considered. 1 mark if one type of hit is considered.]

- For outcome (i), <y, x> gives the physical address.
- For outcome (ii), look-up p^{th} entry in the page table beginning at physical address <y, 0> to get the physical page number.
- For outcome (iii), look-up s^{th} entry in the segment table to get the base address of the relevant page table and then look-up p^{th} entry in that page table to get the physical page number.

COL 216 Computer Architecture : Major Test

Date: 08.04.2022 Time: 14:15-16:15 Max marks: 30

5. (a) Consider two schemes for bus arbitration - (i) daisy chaining and (ii) distributed arbitration by self selection (like NuBus). In (i), the bus grant signal is chained through various masters in priority order. In (ii), IDs of the masters representing their priorities are put on the bus through some logic such that ID of the highest priority requester gets through. Suppose we want the arbitration to take place within one bus clock cycle. How would the cycle time depend on the number of masters in the two schemes? Give justification for your answer.

   (b) A system with multiple disks is required to run programs that are dominated by disk I/O. Data transfer between a disk and memory takes place using DMA over a 32-bit wide synchronous bus with a 500 MHz clock in bursts of length 16. Each transfer requires 2 cycles to initiate (sending address etc), followed by a gap of 20 cycles (due to memory latency) and then 16 cycles for the data burst. (i) What is the peak throughput on the bus possible for such transfers? (ii) How will this figure change if split transactions are supported on the bus?

   (8)

Solution:

(a) Bus arbitration

Let M be the number of masters connected to the bus.

Let N be the number of bits used to define the IDs of the masters (required for NuBus).

Clearly, $N \geq \log_2 M$

In Daisy chain, the bus grant signal has to propagate through the chain of masters. Prapagation delay is proportional to the number of masters, M. Therefore, bus clock cycle time would vary in direct proportion to M.

[2 marks for correct answer. 1 mark if only if arbitration mechanism is described.]

In NuBus, the logic that selects the largest ID among the contending masters has signals flowing from MSB to LSB. Propagation delay here is proportional to the number of bits in IDs, N. Therefore, bus clock cycle time would vary in direct proportion to N.

[2 marks for correct answer. 1 mark if only if arbitration mechanism is described.]

(b) Bus throughput

We ignore the bus acquisition and release time.

Without split transactions

Time for each transfer = 2 + 20 + 16 = 38 cycles [1 mark]

Bytes per transfer = 4 * 16 = 64

Peak throughput on bus = (500 ÷ 38) * 64 = 842.1 MB/s [1 mark]

With split transactions

Time for each transfer = 2 + 16 = 18 cycles [1 mark]

Bytes per transfer = 4 * 16 = 64

Peak throughput on bus = (500 ÷ 18) * 64 = 1777.8 MB/s [1 mark]