IIT JODHPUR

Major Examination: Processor Design (Nov'24) [OPEN-BOOK]

Guidelines (Total time: 180 minutes, Maximum Marks: 45):

- Answer **to-the-point** ONLY. Writing unnecessary statements/lengthy answers may attract penalty.
- **NO clarification is required in any question.** Under **NO** Circumstances, You are permitted to ask any doubt to the invigilator. Assume properly if required and mention that.
- In whichever question, hints are provided, please think wisely to use them in your answer.
- Usage of the internet in any form to seek assistance is **NOT** allowed. Any answer remotely similar to AI/LLM tool-generated text would lead to **NEGATIVE** marks.

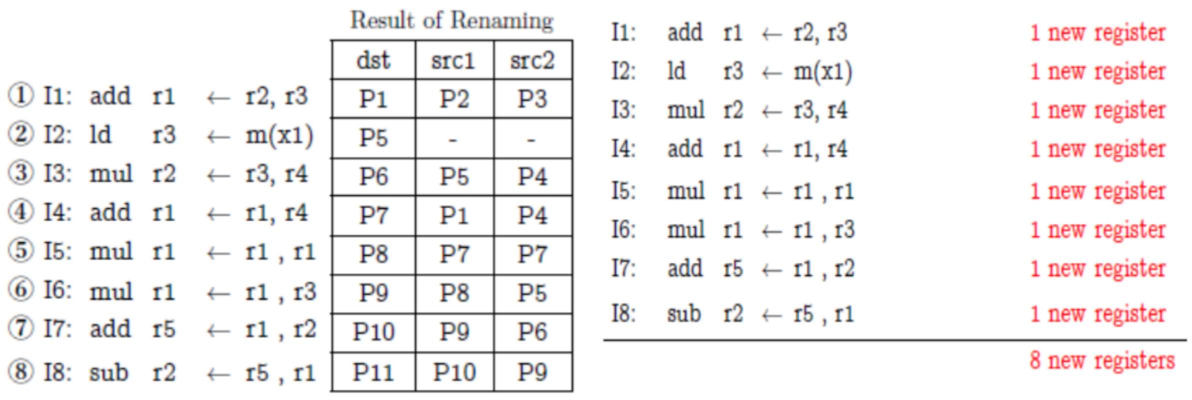
---

1. Suppose it is observed that a function (shown below) is observed to be the main reason behind the sluggish performance of a deep learning application and it is required to redesign the CPU to enable effecient execution of such applications. Because of your detailed knowledge of processor design concepts, you are hired as a consultant to debug this issue by a Noida-based processor design startup, *BharatCPU*. What suggestions would you provide to the designers of *BharatCPU* given that they agreed to pay you INR 10K/hour for your expertise? Why? [3 + 2 marks]

Code	Assembly Output (Instructions)

`for (i=0;i<100;i++){`
`  A[i] = A[i]/2;`
`  }`
	`Loop: LW F2, 0(R1)`
	`      DIV F2, F2, F4      // F4 contains 2`
	`      SW F2, 0(R1)`
	`      ADD R1, R1, 4       // Why 4 is here? Think!`
	`      BNE R1, R3, Loop`

2. Consider that IITJ students develop a processor microarchitecture for a series of CPUs called *MewarProc*. The design of *MewarProc V1* is a simple pipeline core with a special mechanism of register renaming. The problem with conventional register renaming technique is shown in the below figure. To execute the eight instructions in this example, conventional renaming method allocates eight different physical registers, one per instruction. As the number of physical registers in regfile is restricted, the performance of the design becomes a bottleneck because instruction-level parallelism (ILP) becomes limited. **Suggest a technique for implementation in *MewarProc V1* to overcome this problem of the conventional register renaming?** [6 marks]



Result of Renaming

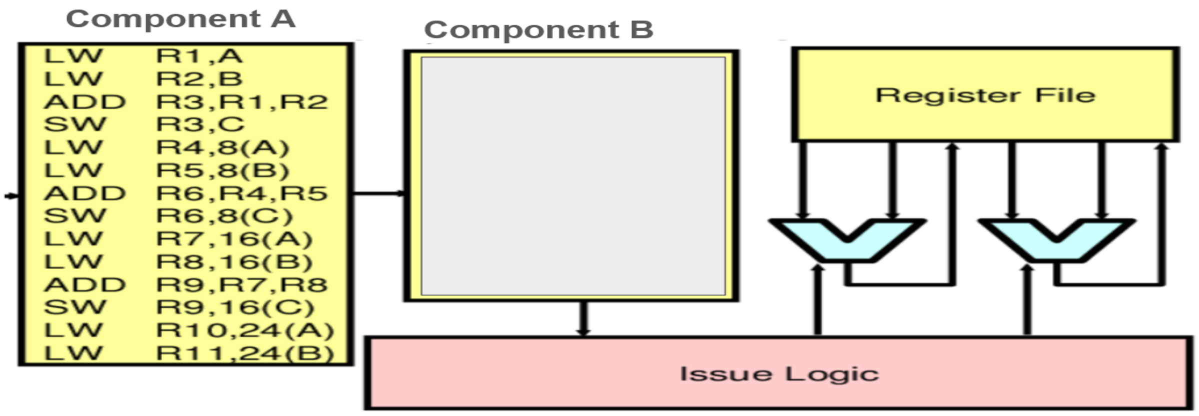
	dst	src1	src2
① I1: add r1 ← r2, r3	P1	P2	P3
② I2: ld r3 ← m(x1)	P5	-	-
③ I3: mul r2 ← r3, r4	P6	P5	P4
④ I4: add r1 ← r1, r4	P7	P1	P4
⑤ I5: mul r1 ← r1 , r1	P8	P7	P7
⑥ I6: mul r1 ← r1 , r3	P9	P8	P5
⑦ I7: add r5 ← r1 , r2	P10	P9	P6
⑧ I8: sub r2 ← r5 , r1	P11	P10	P9

I1: add r1 ← r2, r3	1 new register
I2: ld r3 ← m(x1)	1 new register
I3: mul r2 ← r3, r4	1 new register
I4: add r1 ← r1, r4	1 new register
I5: mul r1 ← r1 , r1	1 new register
I6: mul r1 ← r1 , r3	1 new register
I7: add r5 ← r1 , r2	1 new register
I8: sub r2 ← r5 , r1	1 new register
	8 new registers

Hint: Note that in this example, instructions I1, I4, I5 and I6 form a chain of read-after-write (RAW) dependencies that guarantees that they will be executed in program order and, therefore they will write their results in order in the register file.

3. Consider that the design of *MewarProcV1* is enhanced by IITJ processor design team to obtain *MewarProcV2*. The design of *MewarProcV2* involves a dynamic scheduler to support Out-of-Order(OoO) execution for achieving high performance. Consider the below figure where the internals of the front-end part of *MewarProcV2* is shown and outlined below:

   - Component A: Sequential Instruction Stream
   - Component B: Window of Waiting Instructions
   - Execution Resources: Register File and Issue Logic



```
LW   R1,A
LW   R2,B
ADD  R3,R1,R2
SW   R3,C
LW   R4,8(A)
LW   R5,8(B)
ADD  R6,R4,R5
SW   R6,8(C)
LW   R7,16(A)
LW   R8,16(B)
ADD  R9,R7,R8
SW   R9,16(C)
LW   R10,24(A)
LW   R11,24(B)
```

   Write down the contents of component B (instructions present in component B) initially given the instructions present in component A. Now, assuming the availability of 2 adders (as functional units), re-write the contents of component B (instructions present in component B) after the first round of instruction issue is done (through the issue logic)? [3 + 4 marks]

   Hint: Note that in case you are confused about the instructions shown above, please consider reading the next Question first as meaning of load/store instructions are mentioned there.

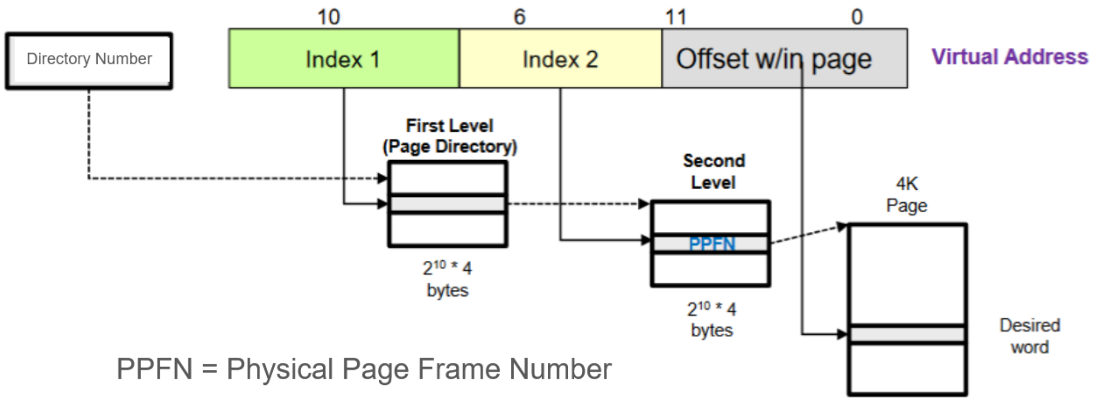
4. Suppose that you are hired as an engineer in a local deep-tech startup *Marudhar Processor Architects (MPA)*. The first design of MPA company is named as *UmaidV1* (after the glorious ex-ruler of Jodhpur estate). You understand that load and store instructions are major culprits of lagging in-order CPU performance. So, you decide to implement a load-store queue (LSQ) that can be integrated in *UmaidV1* and attempt Out-of-Order execution behavior atleast for load/store type instructions. Your design of *UmaidV1* implements simultaneous load bypassing and load forwarding to alleviate the problem due to loads/stores. Consider the ten instructions shown below:

	Instruction	Meaning (SW means store word, LW means load word)
I1	**SW R3, D**	Store contents of R3 into memory-address D
I2	**SW R6, 8(C)**	Store contents of R6 into memory-address [8 + C]
I3	**SW R9, 16(C)**	Store contents of R9 into memory-address [16 + C]
I4	**SW R7, E**	Store contents of R7 into memory-address E
I5	**LW R1, A**	Load contents of memory-address A into R1
I6	**LW R2, B**	Load contents of memory-address B into R2
I7	**LW R4, D**	Load contents of memory-address D into R4
I8	**LW R5, E**	Load contents of memory-address E into R5
I9	**SW R8, F**	Store contents of R8 into memory-address F
I10	**LW R1, F**	Load contents of memory-address F into R1

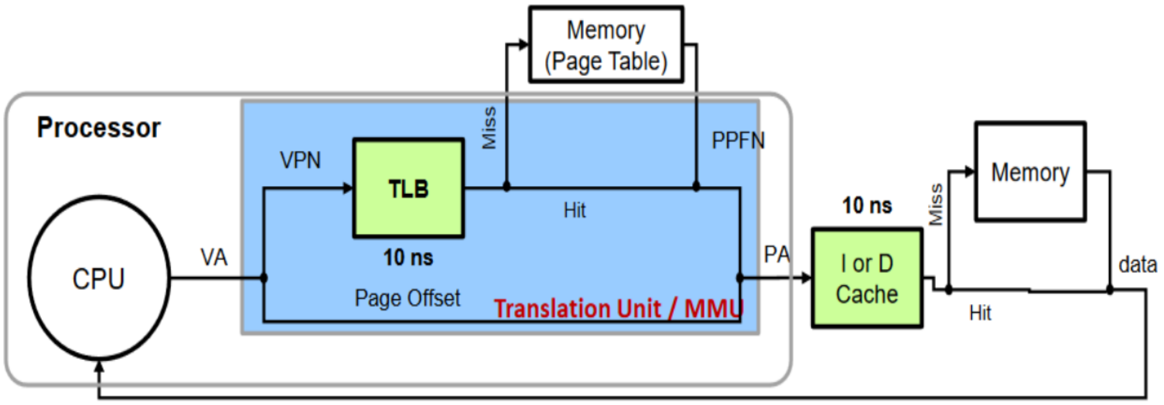
   Explain how the above instruction stream can be optimized by the mcroarchitecture of *UmaidV1*. Devise the controlpath/datapath structure of the LSQ implemented in *UmaidV1*? [3 + 4 marks]

5. Memory access is a major obstacle in speeding up the processor performance. One way to work around this obstacle is to incorporate fast small memories (referred to as cache memories) along with the main memory. The architects of the MPA processor design company (refer previous question) follow this suggestion and come up with a new design code-named as *UmaidV2*. As a designer, you now need to have a mapping strategy between the main memory and the incorporated cache memory. You come up with a scheme as below: If the block number (each block is 16 bytes) in the main memory is odd, put them at position i in cache else put this block at $i + 1$ and continue along this policy for the next pair of blocks. **Design the hardware for this policy implementation and comment on the frequency of eviction of cache blocks if the cache capacity is limited as 4 KB in *UmaidV2* microarchitecture?** [4 + 3 marks]

6. In your job role at the *Marudhar Processor Architects (MPA)* startup, you encounter the architecture of a design named as *MarwarCPU*. This design enabled a 3-level physical-to-virtual memory translation for achieving high performance by providing a larger memory address space. The scheme of this address conversion is shown as below (28-bit virtual address):



You decide to change the above memory setup because 3-level address translation leads to a higher latency ultimately degrading system performance. A new architecture is developed by you and named as *PratapV1* (after the valiant fighter in rajputana history, Maharana Pratap). This design involves having a translation look-aside buffer (referred to as TLB, similar to cache memory). The scheme of *PratapV1* architecture is inspired from the below arrangement:



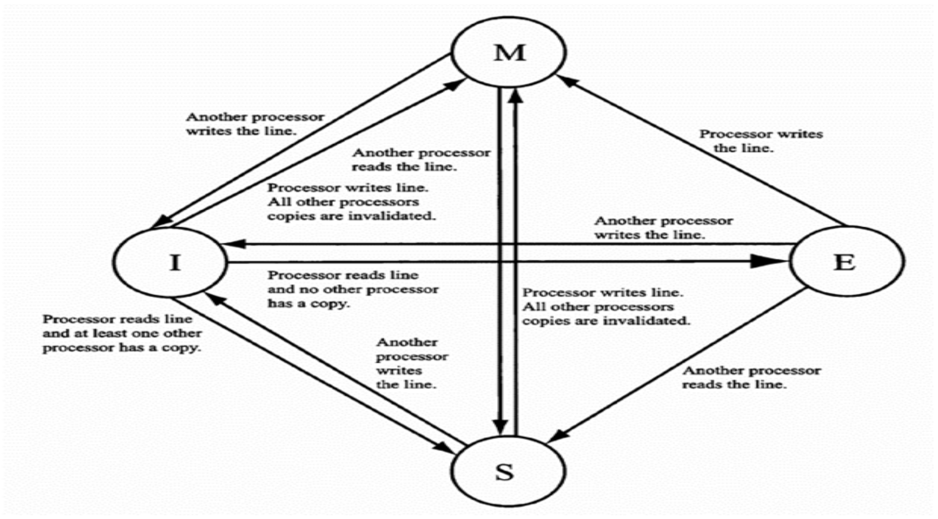
Consider that you are asked to evaluate the performance of your design so that your boss at MPA is convinced and this architecture can be adopted for RTL implementation. **Find out the latency of a memory access in case of a program** *RunMe* **given the below information**:

- There are a total of 8000 integer accesses (memory accesses required) in the sequential program *RunMe*. Each integer is 4 byte in size.

- As the TLB is a small memory and limited in size, the acess of a variable depends on the hit probability. Each page is 4K in size leading to probability of a hit in the TLB as 99.9%. This can roughly translate to as 1 miss every 1K memory/integer accesses.

What is the impact on this latency if the TLB hit probability increases? [3 + 3 marks]

7. As a final solution to increase the system performance, your team at MPA company decides to use the last weapon from the arsenal of architectural techniques, i.e., multi-core design. The choice is made for utilizing four *PratapV1* cores (referred to as P0, P1, P2 and P3) and the multi-core design is named as *PratapV2*. Since multi-core designs involve access of shared memory, your team has decided to utilize the widely popular MESI cache coherence protocol (snoopy version). As a quick recap of this coherence protocol (proposed by Mark Papamarcos under the tutelage of Indian-American computer architect and professor Janak Patel in their seminal paper at the ISCA conference exactly 40 years ago), the below figure is provided for explaining its operation:


M
I
E
S
Another processor writes the line.
Another processor reads the line.
Processor writes line. All other processors copies are invalidated.
Processor writes the line.
Another processor writes the line.
Processor reads line and no other processor has a copy.
Processor writes line. All other processors copies are invalidated.
Processor reads line and at least one other processor has a copy.
Another processor writes the line.
Another processor reads the line.

Consider that state of a cache line for each of the individual cores is maintained in a 8-bit register called as *CohState*. Recall that each processor/core needs two bits for encoding the coherence state. Suppose you fix the encoding as follows: M-01, E-10, S-11, I-00. **Find out the contents of the *CohState* register [assume that coherence state of P0 stored in LSB & coherence state of P3 is stored in MSB] for each of the below sequence of events (assume that each processor starts out with the line containing variable D invalid in their cache)**:

- P0 reads D
- P1 reads D
- P2 reads D
- P3 writes D
- P0 reads D

Your friend who works at Intel suggests that you may try utilizing MESIF coherence protocol (commonly used in Intel's *i*7 series CPUs) where a F state is added (as a specialized form of the S state) and indicates that a cache should act as a designated responder for any requests for the given line. What problem of MESI protocol could be resolved by this enhancement? [5 + 2 marks]