Computer Architecture TDTS10

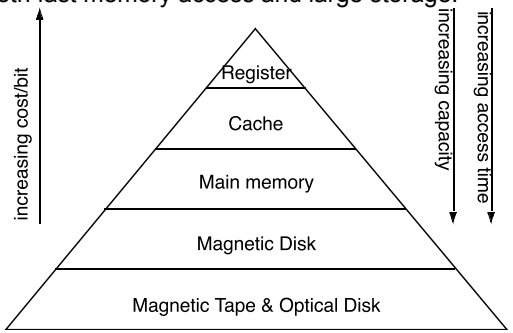
Erik Larsson

Department of Computer Science

LiU expanding reality

Question 1

- Q: Why is the memory system of a computer organized as a hierarchy? What are the basic elements of a memory hierarchy? Illustrate with a figure
- A: The memory system is organized as a hierarchy as one wants both fast memory access and large storage.



Question 2

- Q: Which is the optimal replacement algorithm for a cache memory with direct mapping?
- A: There is no need of a replacement algorithm when you make use of direct mapping

Question 3

- Q: What is the maximal possible memory fragmentation a program itself can cause in a computer system where a 64Kbytes memory makes use of 2Kbytes frames/pages?
- A: First, when paging is used there is no external fragmentation. There is internal fragmentation, and the worst that can happen is that a single byte is needed. Hence, 2Kbytes-1byte is the largest possible fragmentation one can get.

Question 4

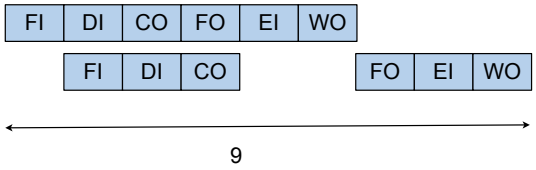
- Q: Assume you have a USB memory stick attached to your computer. After working with the files on the USB memory, a so called friend simply removes the USB memory from the computer. Now, how does your mood depend on the used update policy (write through or write back)?
- A: If my USB is a write-back, data is only updated in the cache and maybe not in the memory. Hence, my mood is towards anger.
- If my USB is a write-through, data is always updated. Hence, my mood is not changed if a so called friend removes the USB.

Question 5

- Q: What is the advantage of separate data and instruction caches when it comes to performance in a non-pipelined architecture?
- A: There is no advantage of having a separate data and instruction cache as there is no memory access conflict when there is no pipeline.

Question 6

- Q: How many cycles would the following sequence of instructions take if they are executed in a 6-stage pipeline (assume one cycle per stage in the pipeline (FI-fetch instruction, DI-decode instruction, CO-calculate operand, FO-fetch operands, EI-execute instruction, WO-write operand)
  ADD R1, (R2)
  ADD R2, R1
- A:



Question 7

- Q: What techniques exist to handle True Data Dependency?
- A: Stall or find alternative instruction

Question 8

- Consider the following sequence:

```
        ADD R1,R2          R1 <- R1 + R2
        SUB R4,R3          R4 <- R4 - R3
        MUL R1,R5          R1 <- R1 * R5
        SUB R1,#2          R1 <- R1 - 2
        BEZ TARGET
        MOVE R1,R6         R1 <- R6
        - - - - - - - - - - - -
TARGET  - - - - - - - - - - - -
```

- Q: Transform this sequence for a machine with delayed branching?

```
                                ADD R1,R2
                                MUL R1,R5
                                SUB R1,#2
                                BEZ TARGET
                                SUB R4,R3
                                MOVE R1,R6
                                - - - - - - - - - - - -
                        TARGET  - - - - - - - - - - - -
```


Question 9

- Q: What is the argument for RISC architecture when it comes to closing the semantic gap between high-level languages and machine code?
- A: Simplicity is a good way to go. Many HLL constructs can be mapped to simple instructions. Only few complex instructions are used, and they can be implemented as a sequence of simple instructions. A simpler instruction set (equal length) makes the CPU simpler; hence it can run faster. A load-store architecture reduces pipe-line penalties. And a simpler architecture makes it possible to squeeze in more registers, which is good as program analysis has shown that:
  - many operands are scalars; hence it is good to have a high number of registers (it speeds up access to have operands in processor)
  - CALL/RETURN is time consuming; a high number of registers can increase performance as registers can be used to store CALL/RETURN information

Question 10

- Assume that the program below is stored in the RAM memory.

```
Address   Instruction/Data
0         LOAD R2, #10              R1=?, R2=10, PC=1
1         LOAD R1, #0               R1=0,        PC=2
2         ADD  R1,(R2)              R1=5,        PC=3
3         BR  5                                  PC=5
4         ADD R1, R2
5         MUL R1, R2                R1=50, R2, 10, PC=6
6         HLT
7         ADD R1, R2
8         SUB R2, #1
9         HLT
10        5
11        8
```

Q: What would the program counter, R1 and R2 contain after execution of the program above?

Question 11

- Q: Is it possible, given the program above, to determine if it is a RISC or CISC processor (motivate)?
- A: It is possible. This is not a RISC machine as there is one instruction like: ADD R1, (R2). Instructions that access memory, which is done by (R2) and also ALU (here through ADD) are not allowed in RISC processors. RISC processors use Load-Store concept. Therefore, this must be a CISC processor.

Question 12

- Q: For a RISC architecture with 16 registers where the instruction length is 12 bits, how many operations is it possible to have in a 2-address instructions scheme?
- A: 16 registers -> 4 bits are needed. If 2-address instruction scheme is used, 4+4 bits are needed. Hence, there are 12-4-4=4 bits left. 4 bits enable 16 operations.

Question 13

- Q: Why make use of demand paging? How does it work?
- A: In demand paging, only pages that are needed are loaded (instead of loading all pages for a program). The advantage is that more programs can be loaded into the memory as each program is only given a few pages.