EE116C/CS151B
Fall 2017

Instructor: Professor Lei He
TA: Yuan Liang

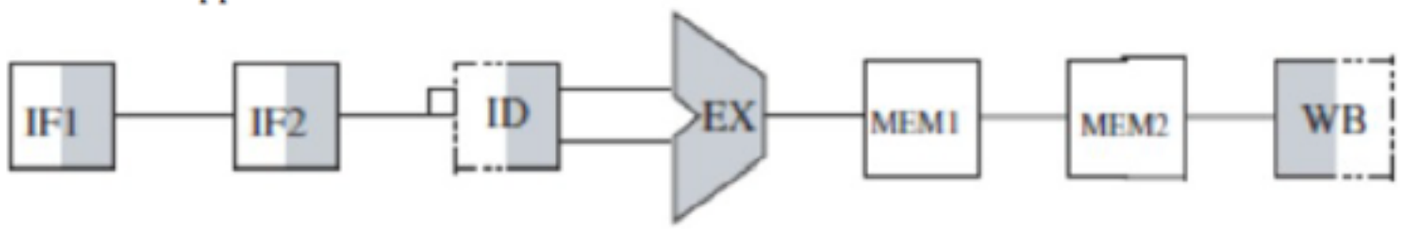
Homework Discussion

Problem 1

The performance advantage of both the multi cycle and the pipelined designs is limited by the longer time required to access memory versus use of the ALU. Suppose the memory access became 2 clock cycles long. Draw the modified pipeline. List all the possible new forwarding situations and all possible new hazards and their length.

Homework Discussion

The modified pipeline is as follows:



We perform the analysis for the lw and R-type instructions. Let the instructions following the lw and R-type instructions be labelled as i1, i2 and i3. Thus, the instruction sequence is lw, i1, i2, i3 where each of the subsequent instructions use the destination register used by lw. Similarly, for the R-type instructions.

The new forwarding and stall situations are:

when defined by lw	when defined by R-type
used in i1 => 2-cycle stall	used in i1 => forward
used in i2 => 1-cycle stall	used in i2 => forward
used in i3 => forward	used in i3 => forward

Homework Discussion

Problem 2

We examine how data dependencies affect execution in the basic five-stage pipeline. Problems in this exercise refer to the following sequence of instructions:

	Instruction Sequence

a.	lw $1, 40($6)
	add $6, $2, $2
	sw $6, 50($1)

b.	lw $5, -16($5)
	sw $5, -16($5)
	add $5, $5, $5

i. Indicate dependencies in the above instruction sequence.

For instruction sequence (a) the hazards are due to dependency on $1 between I1 and I2
dependency on $6 between I2 and I3.
For instruction sequence (b) the hazards are due to dependency on $5 between I1 and I2 dependency on $5 between
$1 and $3

ii. Assume there is no forwarding in this pipelined processor. Indicate hazards and add nop instructions to eliminate them.

a.

```
lw $1,40($6)
add $6,$2,$2
nop
nop
sw $6,50($1)
```

b.

```
lw $5,-16($5)
nop
nop
sw $5,-16($5)
add $5,$5,$5
```

iii. Assuming there is full forwarding, indicate hazards and add nop instructions to eliminate them.

	Instruction sequence

a.	`lw $1,40($6)`
	`add $6,$2,$2`
	`sw $6,50($1)`

b.	`lw $5,-16($5)`
	`nop`
	`sw $5,-16($5)`
	`add $5,$5,$5`

Homework Discussion

Problem 3

In this exercise, we make several assumptions. First, we assume that an N-issue superscalar processor can execute any N instructions in the same cycle, regardless of their types. Second, we assume that every instruction is independently chosen, without regard for the instruction that precedes or follows it. Third, we assume that there are no stalls due to data dependences that no delay slots are used, and that branches execute in the EX stage of the pipeline. Finally, we assume that instructions executed in the program are distributed as follows:

	ALU	Correctly predicted beq	Incorrectly predicted beq	lw	sw
a.	50%	18%	2%	20%	10%
b.	40%	10%	5%	35%	15%

Homework Discussion

a. What is the CPI achieved by a 2-issue static superscalar processor on this program?

	CPI
a.	$0.5 + 0.02 \times 2.5 + 0.98 \times 0.02 \times 2 = 0.589$
b.	$0.5 + 0.05 \times 2.5 + 0.95 \times 0.05 \times 2 = 0.720$

b. In a 2-issue static superscalar processor that only has one register write port, what speedup is achieved by adding a second register write port?

	CPI with 2 register writes per cycle	CPI with 1 register write per cycle	Speed-up
a.	0.589	$0.5 + 0.02 \times 2.5 + 0.98 \times 0.02 \times 2 + 0.70 \times 0.70 \times 1 = 1.079$	1.83
b.	0.720	$0.5 + 0.05 \times 2.5 + 0.95 \times 0.05 \times 2 + 0.75 \times 0.75 \times 1 = 1.283$	1.78

Homework Discussion

c. For a 2-issue static superscalar processor with a classic five-stage pipeline, what speed-up is achieved by making the branch prediction perfect?

	CPI with given branch prediction	CPI with perfect branch prediction	Speed-up
a.	0.589	0.5	1.18
b.	0.720	0.5	1.44

d. Repeat exercise C, but for a 4-issue processor. What conclusion can you draw about the importance of good branch prediction when the issue width of the processor is increased?

	CPI with given branch prediction	CPI with perfect branch prediction	Speed-up
a.	$0.25 + 0.02 \times 2.75 + 0.98 \times 0.02 \times 2.5 + 0.98^2 \times 0.02 \times 2.25 + 0.98^3 \times 0.02 \times 2 = 0.435$	0.25	1.74
b.	$0.25 + 0.05 \times 2.75 + 0.95 \times 0.05 \times 2.5 + 0.95^2 \times 0.05 \times 2.25 + 0.95^3 \times 0.05 \times 2 = 0.694$	0.25	2.77

Review

- why virtual memory
- how virtual memory works
  - map to mem
  - map to disk
  - replace on page faults

Review

- **why virtual memory**

  - **Virtual address space: addresses 0 to 2^{32}–1**
    - **(or 0 to 2^{64}–1 on machines with 64-bit addressing).**
    - **even if there is much less physical memory.**
  - **Virtual memory also provides protection**
    - **One process can't corrupt the data of another**
    - **Another user, running on the same machine, can't see your data**

Review

How VM differs from memory caches

- **MUCH higher miss penalty (millions of cycles)!**
  - **Therefore:**
    - **large pages [analogous to cache line] (4 KB to MBs)**
    - **associative mapping of pages (typically fully associative)**
    - **software handling of misses (but HW handles hits!!)**
    - **write-through never used, only write-back**

space

- **Many pages**
  - **With 64 MByte memory, 4KBytes/page, have 16K pages**
    - **It's not practical to have 16K comparators**
    - **Nor to have software do many comparisons**
  - **How can we get virtual memory with full associativity???**

multi-program

Review

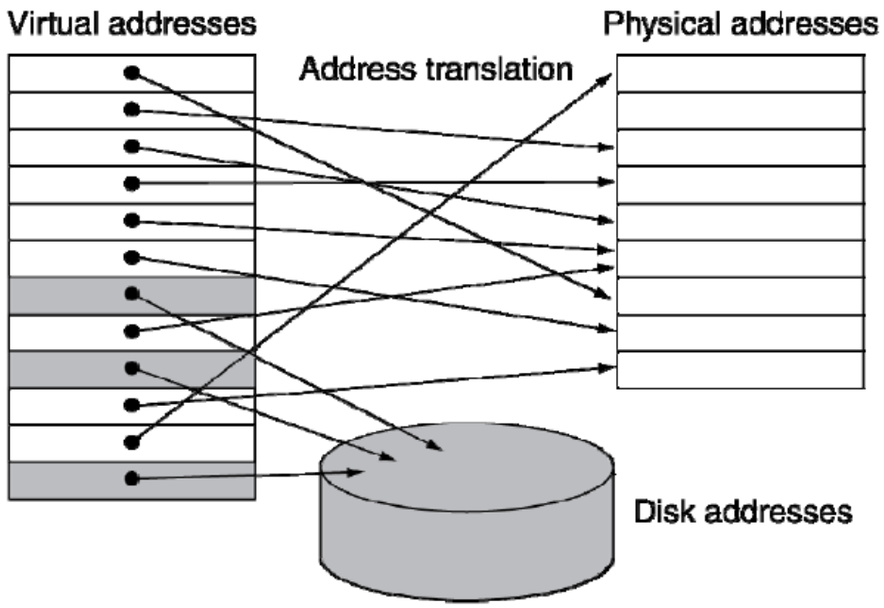
- **Problems to overcome**
  - **1. how to allocate data on memory.**
    - in cache: we do hashing (direct mapping / set associative)
    - in memory: we do full associative
    - why?
  - **2. how to tell the location of data on disk/memory**
    - in cache: we do hashing + tag
    - in memory: we use lookup table

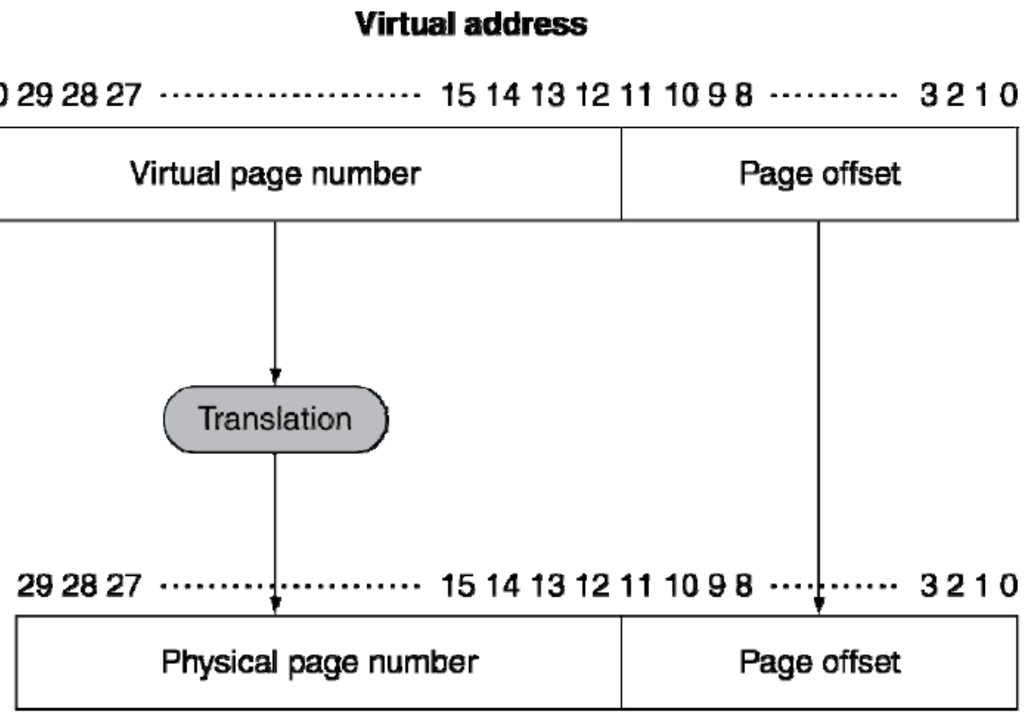
Review

- **Problems to overcome**
  - **1. address translation**
    - a table
  - **2. data locating**
    - 1) whole mem search
    - 2) partial mapping table to record
    - 3) full mapping table to record
      - save time
      - reduce missing rate

Review

- **how virtual memory works**





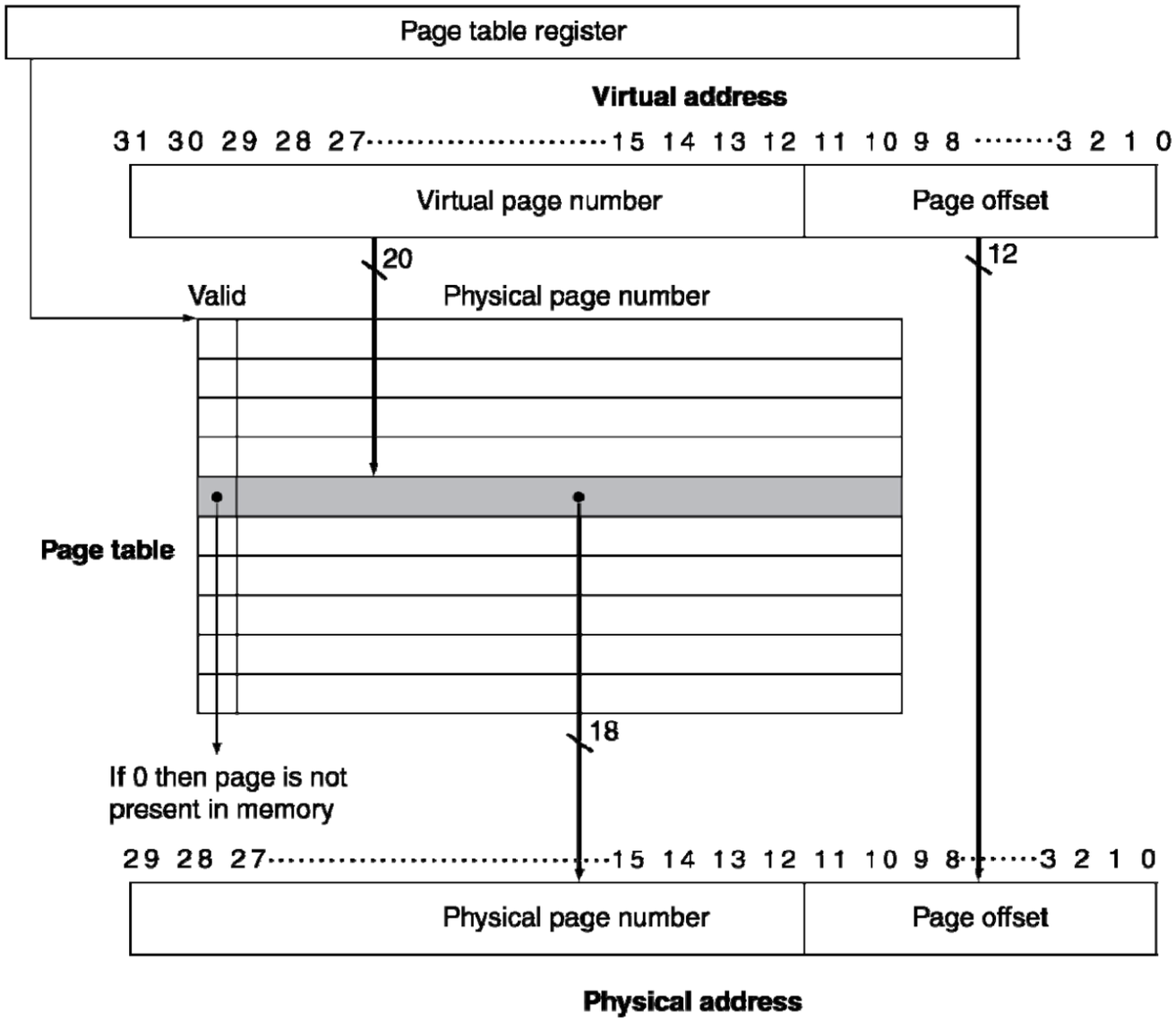
Review

- **how virtual memory works - ground truth!**
  - VM address space is larger than real main mem space. (illustration)
  - VM is divided into **fix-sized page** (for easy management).
    - Fully associative placement. (less miss/or say **page fault** here)
    - Write-back.
- **As On page fault, the page must be fetched from disk.**
- **Takes millions of clock cycles.**

Review

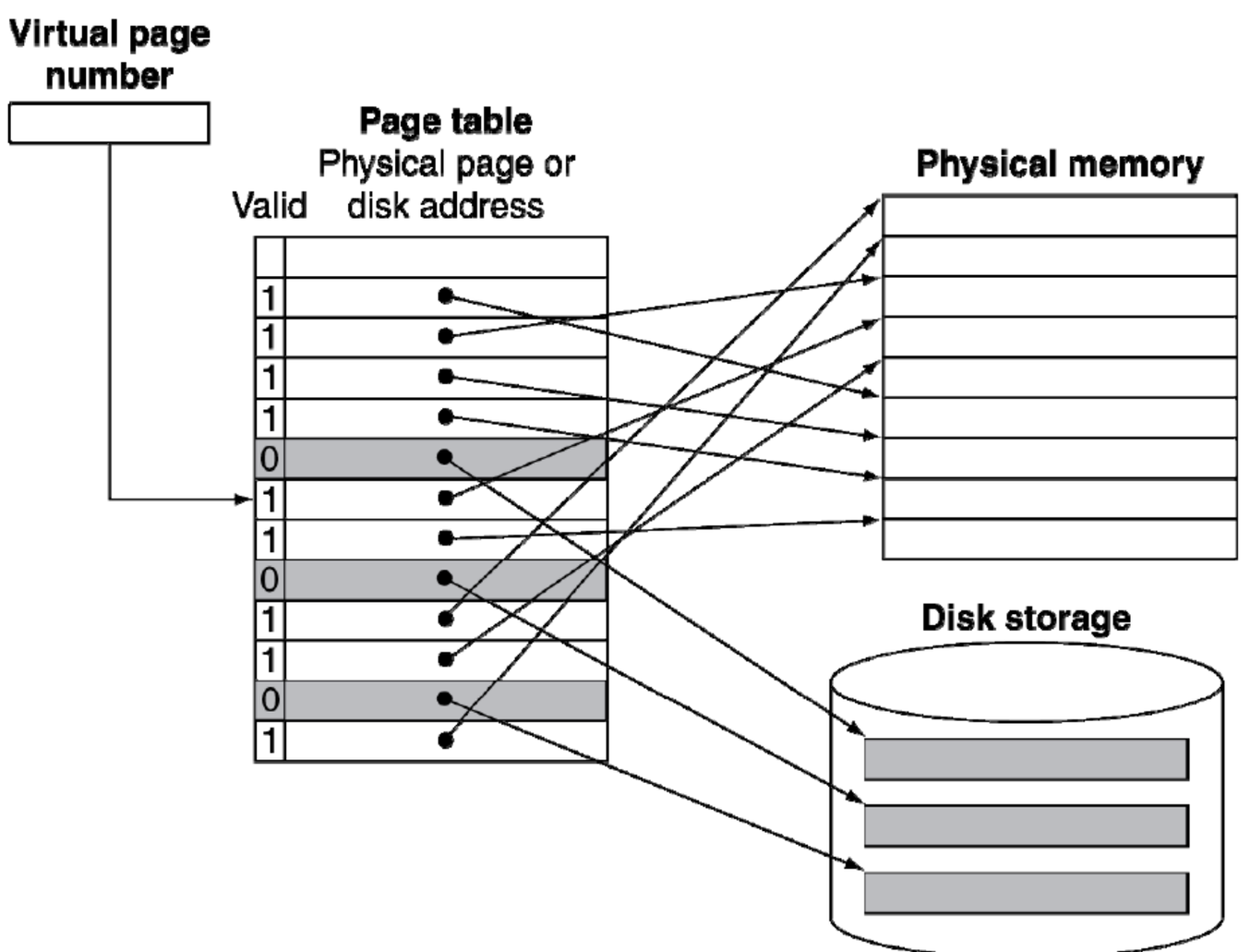
- **how virtual memory works**
  - **Each program has its own page table. (Resides in cache/mem)**
  - **Page table register (Hardware) points to the start of the table.**

19 bits table:
rounded to 32 bits:
1. for ease
2. other info.
(protection, dirty, etc.)



Review

- **how virtual memory works**
  - **How to access physical mem**
  - **How to access physical disk**
  - **Swap space (on disk) allocated for each program**
- **Replace policy:**
  - **LRU**
  - **approximate LRU**



Review

- **how virtual memory works**
  - Each program has its own page table. (Resides in cache/mem)
  - Why cache/mem?
    - **Example: 512 MB virtual addresses space, 4 KB pages.**
      - **Then we need 2^{29} / 2^{12} = 2^{17} entries in the page table.**
    - **Suppose each entry is 4 Bytes.**
      - **Then page table takes 2^{19} Bytes (a half megabyte)**
      - **... and each process needs its own page table.**
    - **So part of page table may not even be in cache.**
  - Is it wasting time a lot? go to mem, read table, fetch data from mem to cache.
    - Yes, it is. But it is necessary. read table + read data ~= 2 * read data
  - How can we reduce this time complexity?
    - Translation look-aside buffer (TLB)

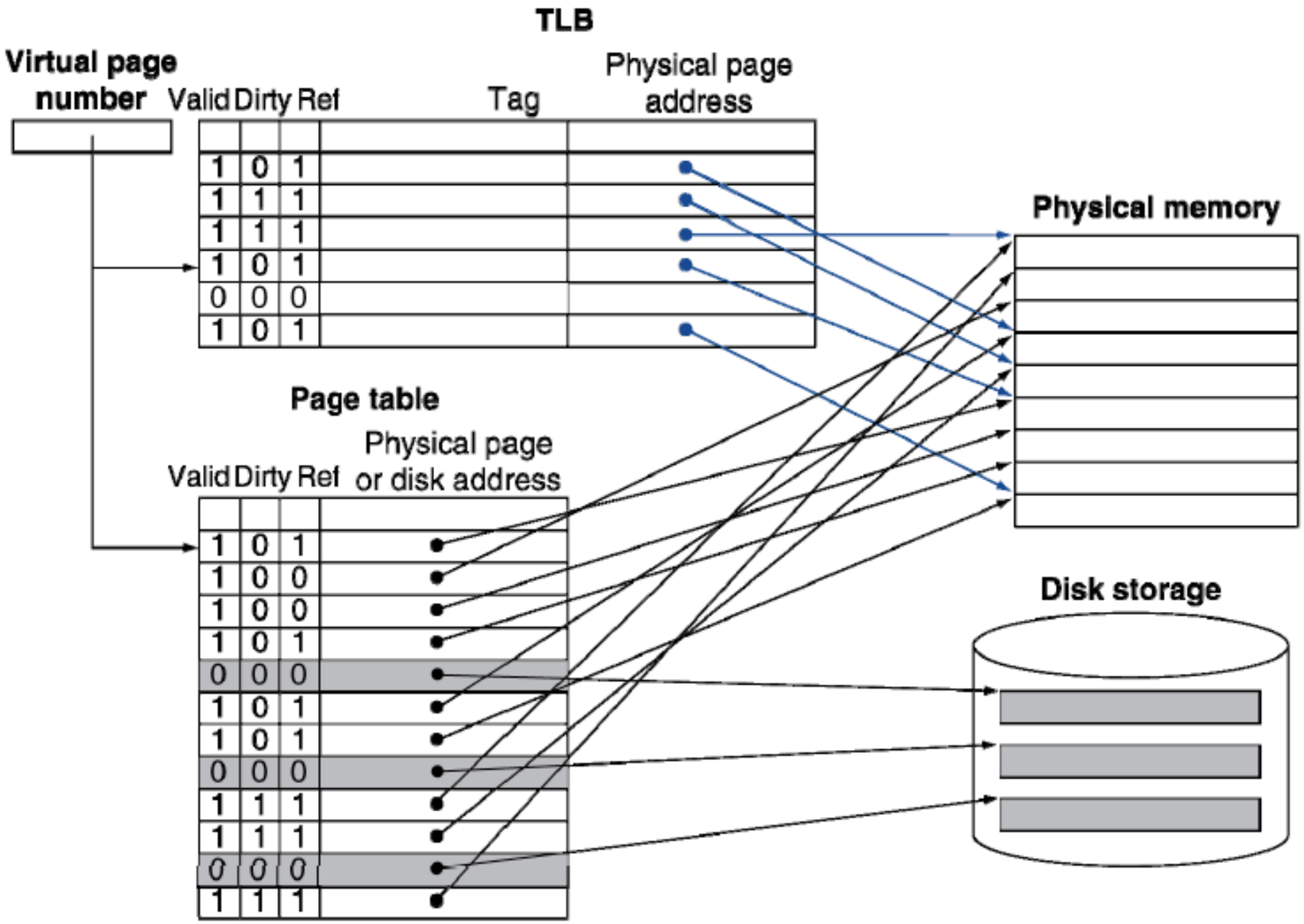
Review

- **how to reduce virtual memory overhead**
  - **Translation Look-aside Buffer (TLB) for Fast Address Translation**

- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software

Review

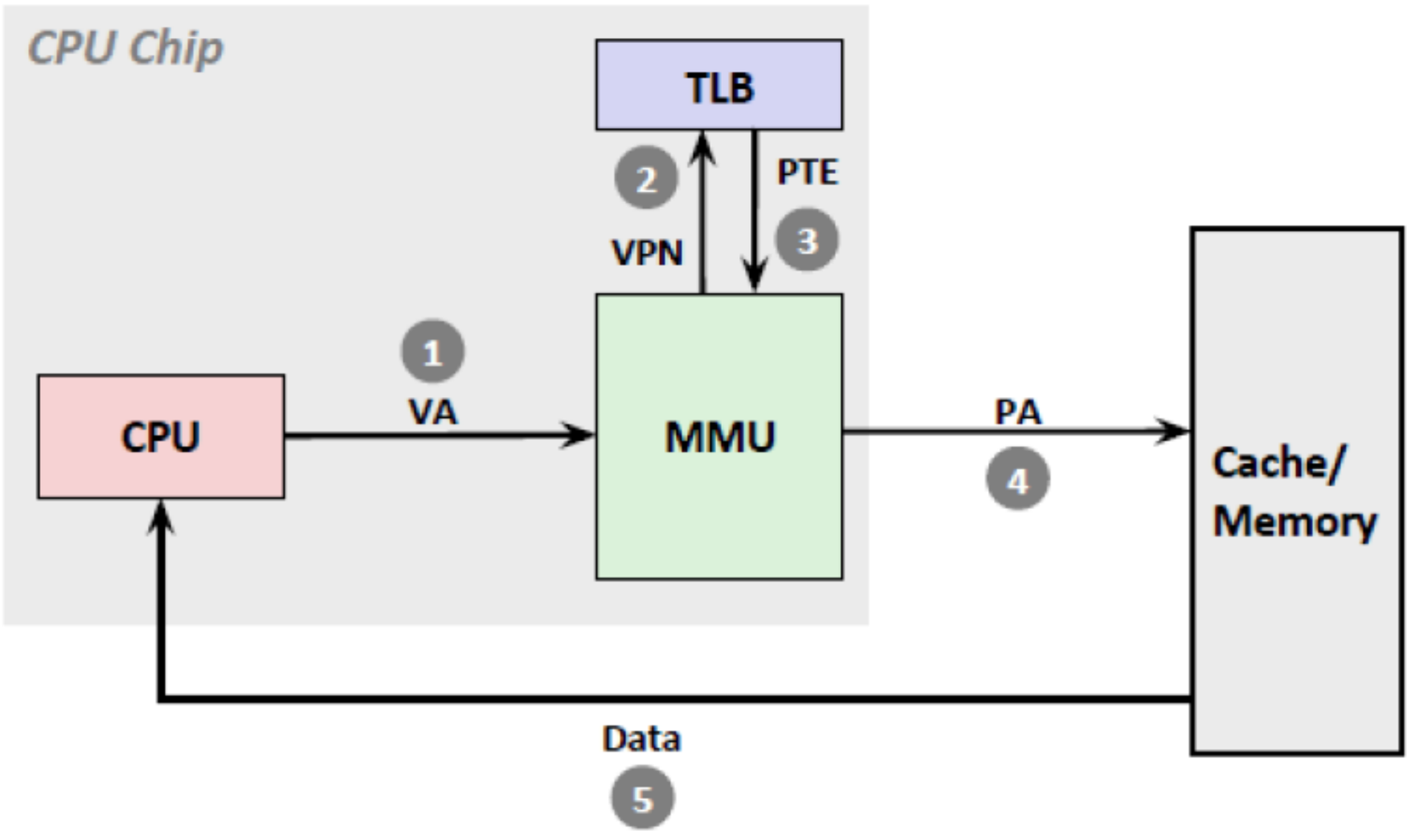
- **how to reduce virtual memory overhead**
  - **Translation Look-aside Buffer (TLB) for Fast Address Translation**
  - **within CPU**



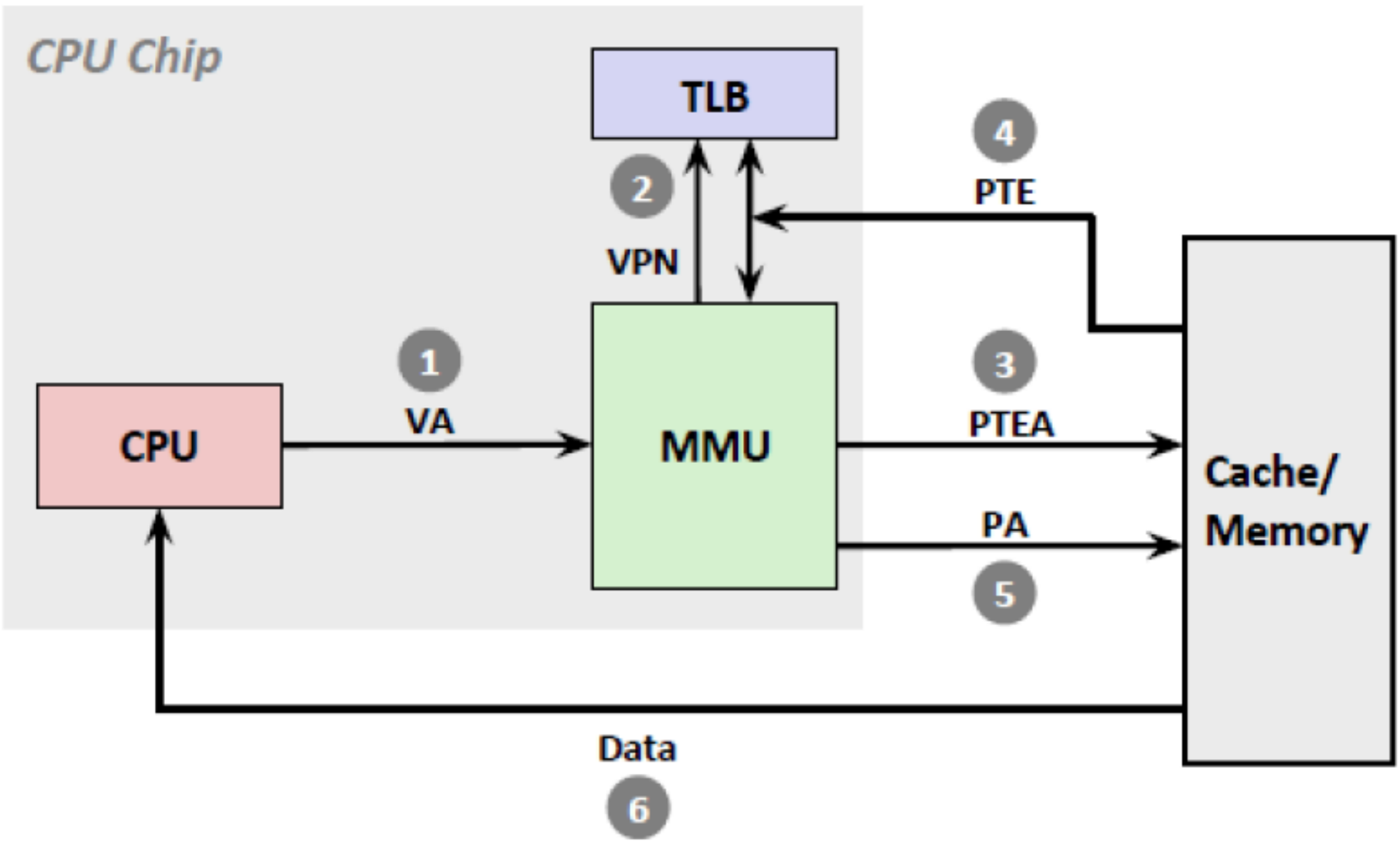
Review

- **how to reduce virtual memory overhead**
  - **Translation Look-aside Buffer (TLB) for Fast Address Translation**
  - **within CPU**
    - If page is in memory
      - Load the PTE from memory and retry
      - Could be handled in hardware
        - Can get complex for more complicated page table structures
      - Or in software
        - Raise a special exception, with optimized handler
    - If page is not in memory (page fault)
      - OS handles fetching the page and updating the page table
      - Then restart the faulting instruction

Review



Review


CPU Chip
TLB
2
VPN
4
PTE
1
VA
CPU
MMU
3
PTEA
PA
5
Cache/
Memory
Data
6

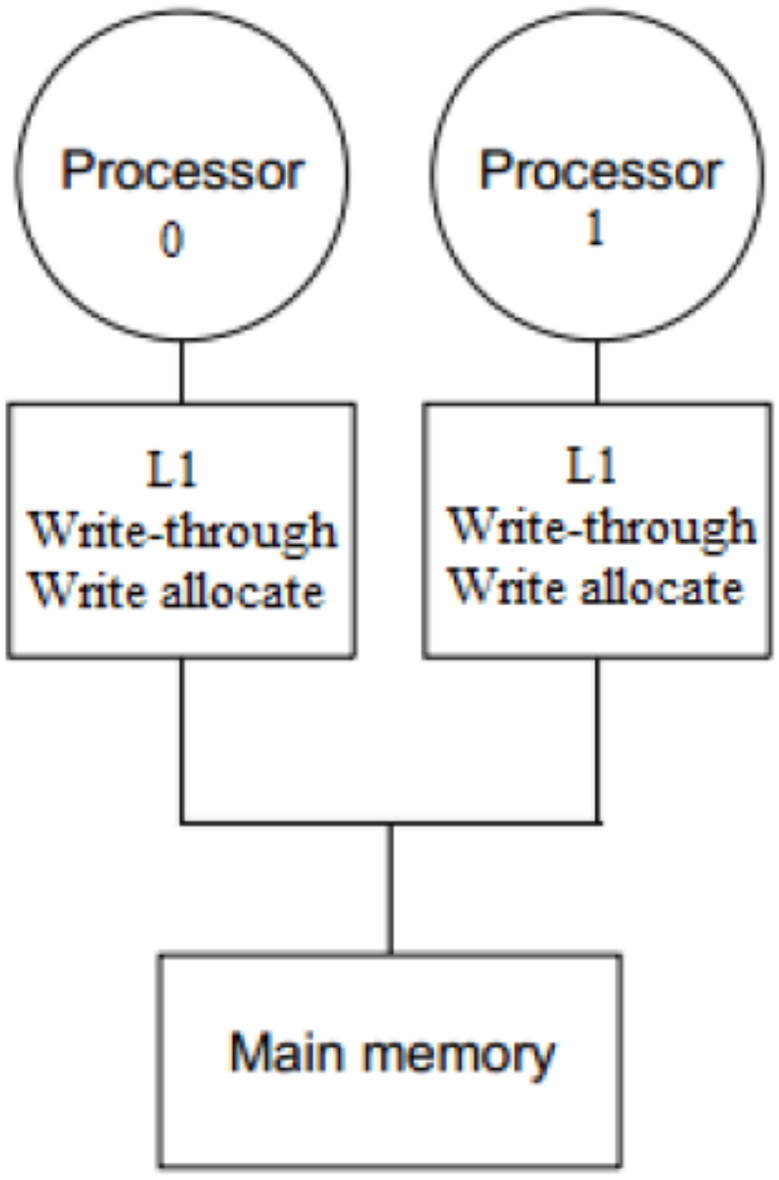
Review

- **Some of Trade-offs:**

Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

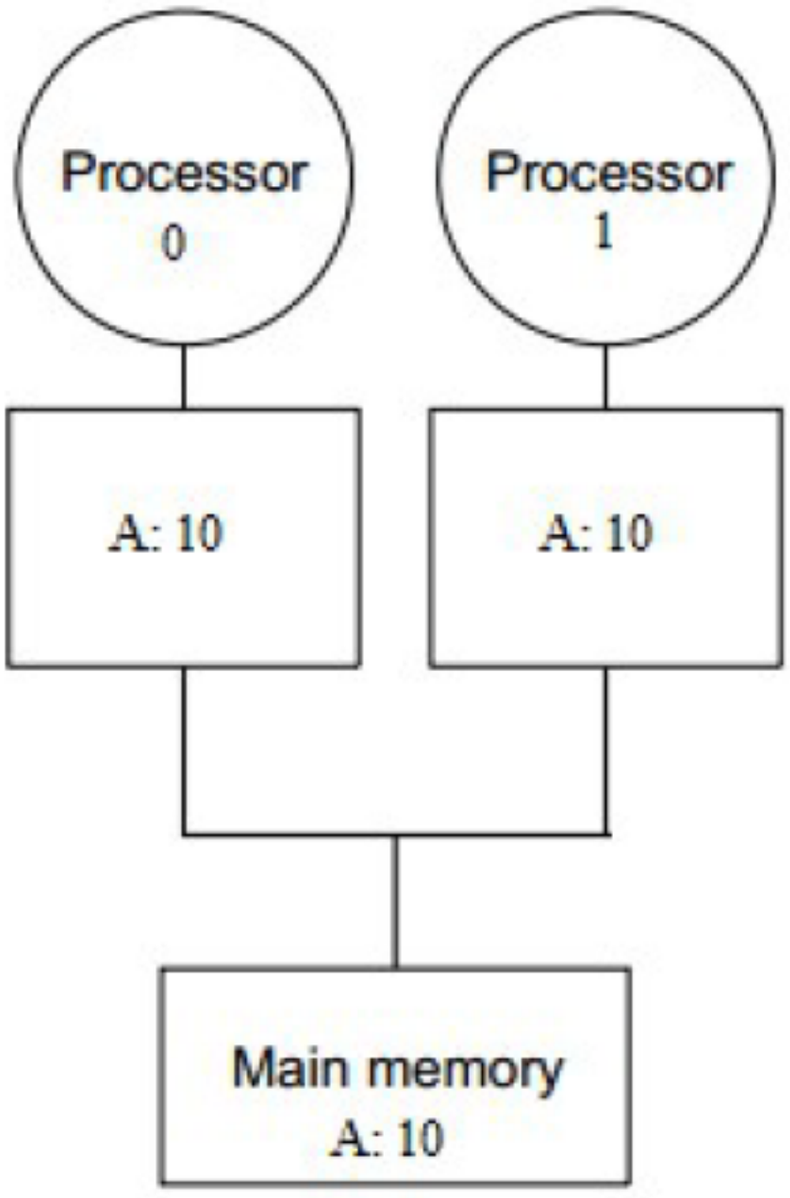
Sample question 1




```
P0:
addi $t0, $zero, 10
sw $t0, A
P1:
lw $t0, A
```

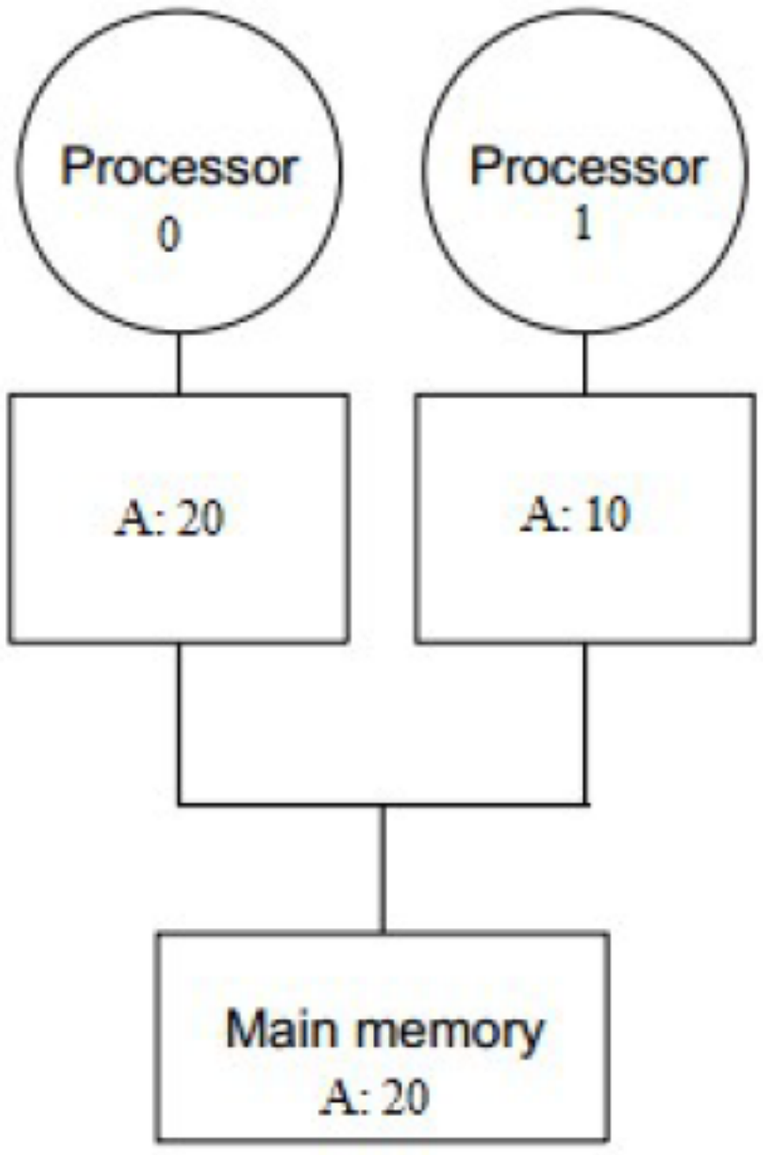
```
P0:
addi $t0, $zero, 20
sw $t0, A
P1:
lw $t0, A
```

Sample question 1




```
P0:
addi $t0, $zero, 10
sw $t0, A
P1:
lw $t0, A
```

Sample question 1




```
P0:
addi $t0, $zero, 20
sw $t0, A
P1:
lw $t0, A
```