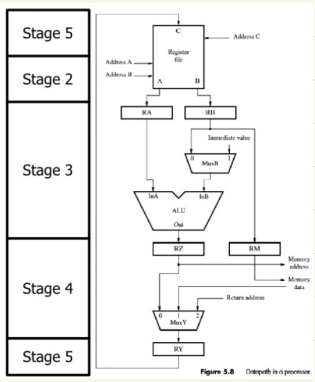
Lecture 9 – Basic Processing Unit (BPU)

Component structure

mem.	CISC	RISC
add.	MAR	R2
data	MDR	RM

CISC
→ Components connected many-to-many
→ flexible

RISC
→ Components connected in sequence
→ pipelined by design

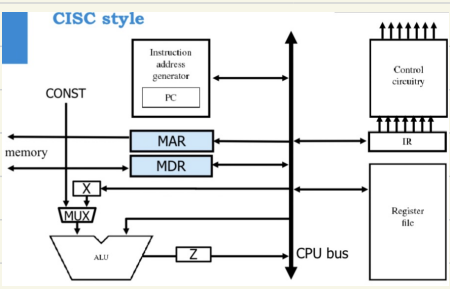


CPU (RISC):

1. **Fetch** instruction from memory
2. **Decode** instruction and read registers
3. **Execute** ALU operation
4. **Memory** access (if needed)
5. **Writeback** to register (if needed)

CPU (CISC):

1. Instruction fetch
   1) Instruction address gen.
   2) MAR + MDR
   3) IR
2. Operand fetch
   1) IR → instruction + operands
   2) values in X, MUX, or directly
3. Instruction execution
   1) ALU output in Z
5. Write back
   1) output in register file

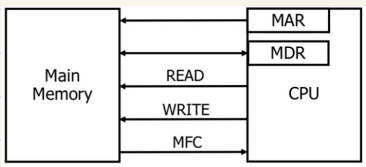


Quiz 9.1.

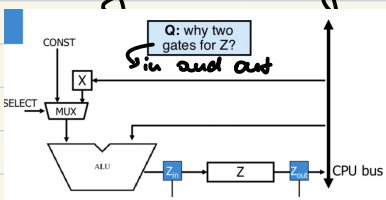
1. inst. fetch → IR, MAR, MDR, PC
2. writing data to memory
   → address moved to MAR
3. writing data to memory
   RISC → RM

Important Concepts

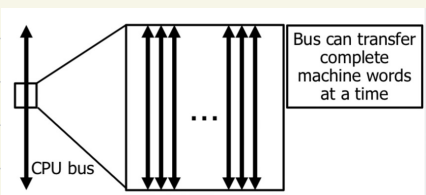
Interacting with main memory



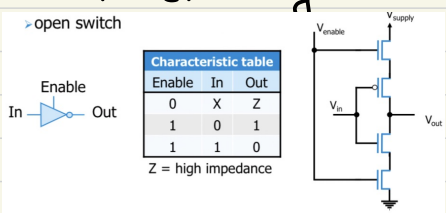
Register Gating



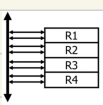
CPU bus



Tri-state gates



Register File

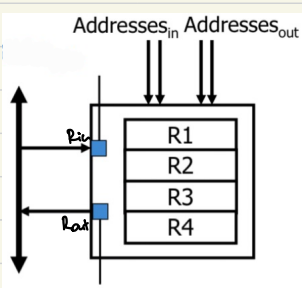


→ Gates R_{in} and R_{out}

→ $Addresses_{in}$ and $Addresses_{out}$

2^x registers → $\log_2 2^x = x$ - bit addressing

$\Rightarrow 2^{(x+1)}$ - bit register file addressing



ADD R2, R1 ↔ R1 + = R2

Fetch

1. PCout, MARin, READ, WMFC → wait memory to fully complete (always after READ and WRITE)
2. MDRout, IRin

Execute

3. R2out, Xin
4. R1out, ALUadd, Zin
5. Zout, R1in

Quiz 9.2 ADD R2, (R1)

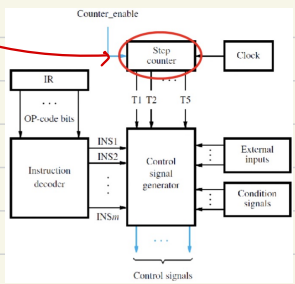
4. R1out, MARin, READ, WMFC
5. MDRout, ALUadd, Zin
6. Zout, MDRin, WRITE, WMFC

Lecture 10 – Basic Processing Unit (BPU)

STORE R2, (R1)

1. PCout, MARin, READ, WMFC
2. MDRout, IRin
3. R1out, MARin
4. R2out, MDRin, WRITE, WMFC

	PC_{out}	PC_{in}	$R1_{in}$	$R1_{out}$	MAR_{in}	MDR_{out}	IR_{in}	...	WMFC
⇨1	1	0	0	0	1	0	0		1
2	0	0	0	0	0	1	1		0
3	0	0	0	1	1	0	0		0
4	...								
5	...								
6	...								
7	...								



control signals:
- instruction code
- control step counter
- flags & external signals

Quiz 10.1

32 general purpose registers
42 two-operand functions in ALU (X reg for input, 2 for output)
MDR and MAR registers
32 → $\log_2 32$ = 5-bit register addressing
2(5+1) = 12-bit register file addressing
42 < 64 → 6-bit ALU operation addressing
X reg and MAR (only in) → 1 bit
2 reg and MDR (in and out) → 2 bits
12 + 6 + 1 + 2 + 1 + 2 = 24 bits

ADD R1, R2 → fetch = 2 steps } 5 steps
(no memory) → execute = 3 steps }

SUB #24(R3), R0 → fetch = 2 steps
(1 memory) → execute = 4 steps } 7 steps
displacement = 1 step

Method 1 – Hardwired Control

RFwrite = T5. (ALU + Load + Call)

→ fast

→ inflexible

Quiz 10.2

vertical: 8 reg, MAR+MDR, 4 ALU

8 bits = ↓3 + ↓3 + ↓2

microprogrammed — flexible — complex hardware

hardwired — complex design — higher performance

Method 2 – Microprogrammed Control

μ-program memory — control word=μ-instruction

	PC_{out}	PC_{in}	$R1_{in}$	$R1_{out}$	MAR_{in}	MDR_{out}	IR_{in}	...	WMFC
1	1	0	0	0	1	0	0		1
2	0	0	0	0	0	1	1		0
3	0	0	0	1	1	0	0		0
4	...								
5	...								
6	...								
7	...								

- Increased flexibility (branching)

Instruction

MOV	R1	A
ADD	R1	R2

PC_{out}	PC_{in}	IR_{in}	...	END
PC_{out}	PC_{in}	IR_{in}	...	END
PC_{out}	PC_{in}	IR_{in}	...	END

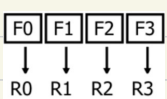
microinstruction

compared to hardwired: branching vs. sequencing

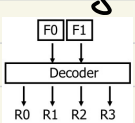
compared to full CPU:
→ no ALU, registers, ...
→ response to signals

Micro-instruction layout

1. No/little encoding: horizontal organization
   1) large words
   2) less decoding
   3) faster



2. Much encoding: vertical organization
   1) small words
   2) more decoding
   3) slower



3. Mixed

micro-instruction	PC_{out}	IR_{in}	MAR_{in}	MDR_{out}	REG_{in} addr
1	1	0	1	0	00
2	0	1	0	1	00
3	0	0	0	0	10
..					
..					

Horizontal organization — vertical organization

F1	F2	F3	F4	F5	F6	F7	F8

Field 1 (4 bits): Register address_in
Field 2 (4 bits): Register address_out
Field 3 (2 bits): RF_in/out
Field 4 (4 bits): Function ALU
Field 5 (2 bit): Read/Write/Nop
Field 6 (1 bit): Carry-in ALU
Field 7 (1 bit): WMFC
Field 8 (1 bit): End

– flexible
– slower
– more complex to build in hardware

Microroutine example
JN PC+OFFSET

Address	Micro-instruction	
0	PC_out, MAR_in, Read, WMFC	Fetch Instruction
1	MDR_out, IR_in	
2	Branch to starting address routine (here, 103)	Decode
103	PC_out, X_in, if N=0 then goto address 0	← Test N bit
104	IR-Offset_out, MUX_x, F_alu = "ADD", Z_in	← Add offset
105	Z_out, PC_in, End	← Update PC

Microroutine example
ADD R1, R2

Address	Micro-instruction	
0	PC_out, MAR_in, Read, WMFC	Fetch Instruction
1	MDR_out, IR_in	
2	Branch to starting address routine (here, 42)	Decode
42	RF_read, RF_addr_out = 1, X_in	Exec
43	RF_read, RF_addr_out = 2, MUX_x, F_alu = "ADD", Z_in	
44	Z_out, RF_write, RF_addr_in = 2, End	

Self-Study 6

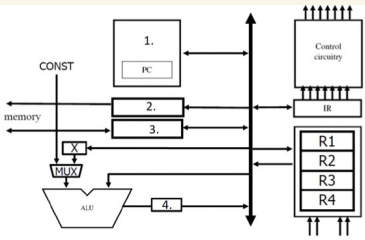


Figure 1: Basic Processing Unit with missing components.

1. (5 mins) Figure 1 shows the components of a CISC-style basic processing unit. Name the numbered components and describe what kind of information is stored in them.

#	name	description
1	instruction address generator	generates the address of the next instruction
2	MAR	Memory Address Register – stores memory address
3	MDR	Memory Data Register – stores data from memory
4	Z	stores the result of ALU operations

2. (5 mins) Write down the 5 stages of a RISC-based BPU and explain what is happening during each of them:

1. Fetch – get instruction from memory
2. Decode – evaluate the OP code and the encoded register addressing
3. Execute – perform arithmetic and logic operations in the ALU
4. Memory – access information from memory if needed
5. Writeback – communicate back to registers if needed

16 registers $= 2^4 \rightarrow 2(4+1) = 10$ bits ($< 12 = 2^3$)

ALU with 104 instructions $\rightarrow$ 7 bits

X as input (only in) $\longrightarrow$ 1 bit

2 as output (in and out) $\rightarrow$ 2 bits

20 bits

Instruction Fetching:
1. PCout, MARin, READ, WMFC
2. MDRout, IRin

ADD R2, (R4), R3 => R3=R2+(R4)
1. R2out, Xin
2. R4out, MARin, READ, WMFC
3. MDRout, ALUadd, Zin
4. Zout, R3in

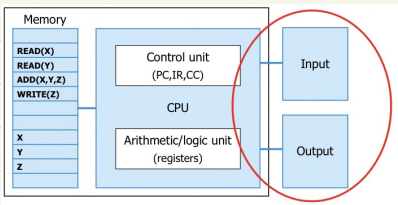
DIVIDE (R1), (R2), R3
1. R1out, MARin, READ, WMFC
2. MDRout, Xin
3. R2out, MARin, READ, WMFC
4. MDRout, ALUdivide, Zin
5. Zout, R3in

BRANCH PC + (R3)
1. R3out, MARin, READ, WMFC
2. MDRout, Xin
3. PCout, Xout, ALUadd, Zin
4. Zout, PCin

encoding	vertical	horizontal
register file with 32 registers	$2(5+1) = 12$	$2(32+1) = 66$
MAR+MDR	3	3
20 ALU operations	5	20
=> X register	1	1
+ 2 register	2	2
+ MUX	1	1
	24 bits	93 bits

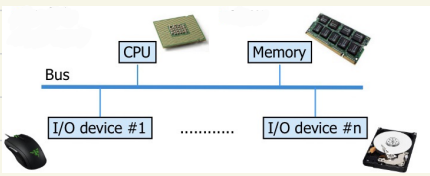
Lecture 11 – I/O

Large variety of devices
- → size
- → speed
- → distance

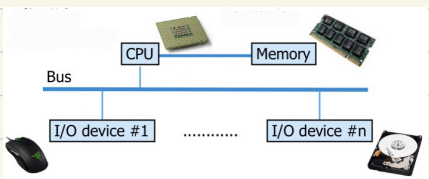


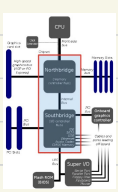
Buses and Interfaces

Singe-bus architecture
- Pro: simple, flexible, low cost
- Con: not scalable, slow



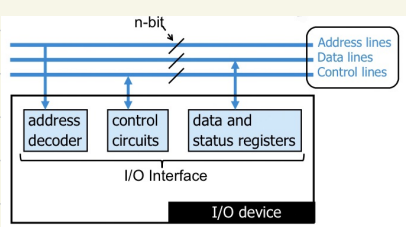
Dual-bus architecture
- Pro: fast, flexible, quite cheap
- Con: not scalable, complex





Typical motherboard
- Northbridge → high performance
- Southbridge → flexibility
- Chipsets → AMD AM5 (2022 –)
- → Intel Coffee lake (2017 –)

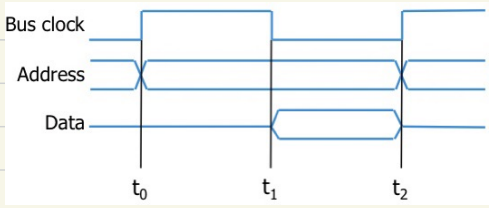
Bus Interface



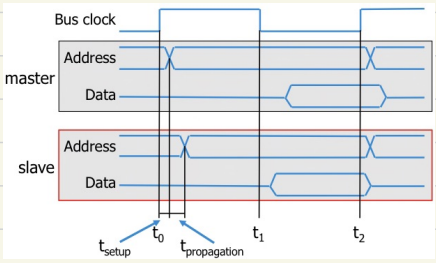
address decoder – detect if data is for device
data registers – to store/buffer in/out data
status and control registers
- → to certify status of device
- → to control transfer

Synchronous bus

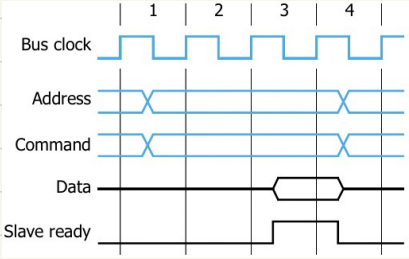
ideal case
→ perfect synchronization across all devices
→ instantaneous transitions



Propagation delays



Multi-cycle bus



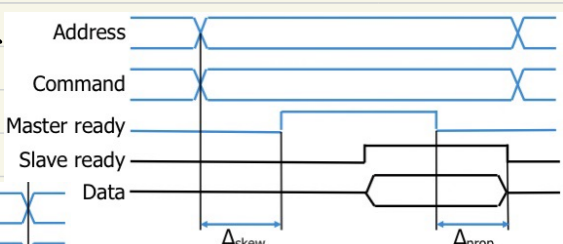
bus driver delay	1.5 ns
max bus length	15 cm
address decoder delay	6 ns
time to fetch data	3 – 15 ns
setup time (control circuitry)	≤ 2 ns

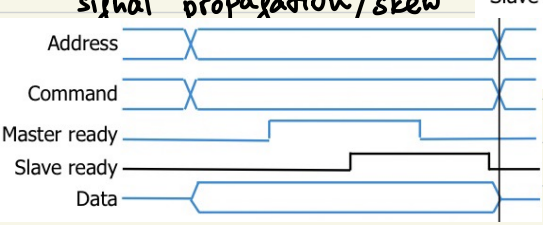
$$t_{max} = \frac{l}{v} = \frac{15\,cm}{\frac{2}{3}c} = \frac{1.5 \times 10^{-2}\,m}{2 \times 10^{8}\,\frac{m}{s}} = 7.5 \times 10^{-11}\,s = 75\,ns$$

Asynchronous bus

→ Explicit handshaking
→ timing must account for signal propagation/skew

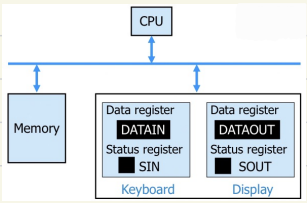
input operation



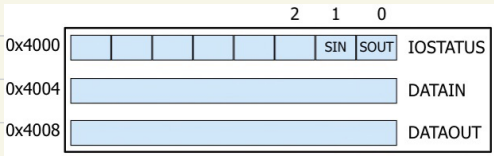


output operation

Program-controlled I/O



assigning address range

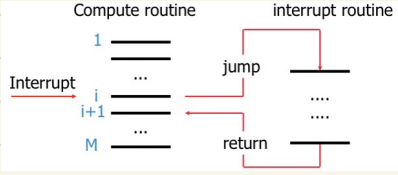


Unconditional I/O – no synchronization with I/O device

Passive signaling (Polling) – synchronization between CPU and I/O device by programmed interrogation by CPU

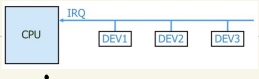
Active signaling (Interrupts) – synchronization between CPU and I/O device by active interrupt of device

Interrupts



ISR – Interrupt Service Routine

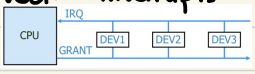
1. I/O device alerts CPU by hardware signal IRQ – Interrupt Request Line
2. CPU stops program execution
3. Interrupts are disabled
4. Device is informed of acceptance and clears IRQ
5. ISR is invoked to handle device's request
6. Interrupts are enabled
7. Execution of program resumes



shared IRQ line

CPU polls status registers

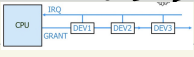
vector interrupts



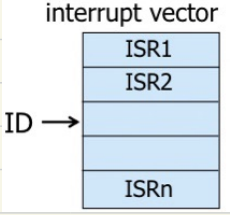
GRANT signal from CPU

Device sends ID on data bus

CPU calls ISR from interrupt vector [ID]



Rule out simultaneous interrupts by prioritizing devices fixed order



Lecture 12 – DMA & Memory Organization

Non-programmed I/O

A separate active entity executes the I/O tasks.

→ User instructs the device about what to copy

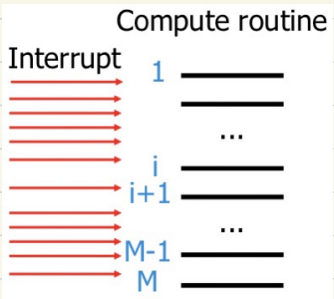
→ Device takes care of data transport

Direct Memory Access (DMA)

1 instruction: MOVE (= COPY)

Special I/O processors

- more versatile
- run their own program



DMA controller

→ independent entity

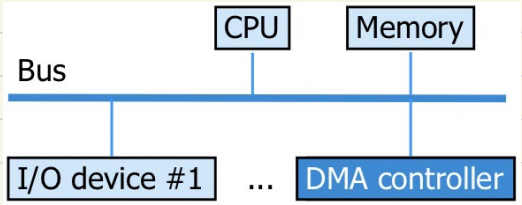
→ specialized in data transfer

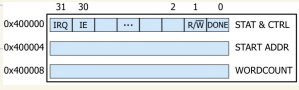
→ block based

Bus priority

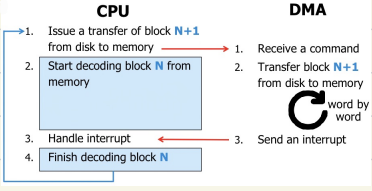
→ cycle stealing

→ burst mode





CPU ↔ DMA interactions



DMA controller

- Access as a normal I/O device
  → signals when DONE and/or by interrupt transferring data
- Acts as a bus master when
  → sets addresses and controls lines

32-bit RISC machine

8 registers

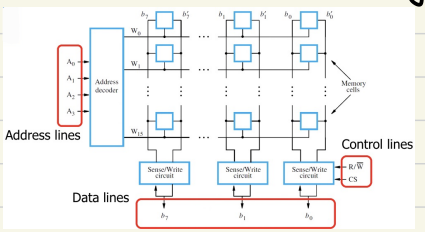
blockmove (src, dst, n)

R1 = src | R2 = dst | R3 = n

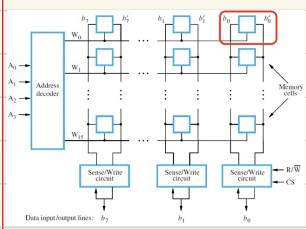
```
MOV (R1), R4
MOV R4, (R2)
SUB $1, R3
ADD $4, R1
ADD $4, R2
```

DMA without caching
→ every instruction needs to be fetched from memory
→ bus is mostly occupied

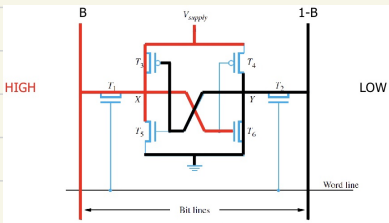
DMA with cache
→ bus is mostly free



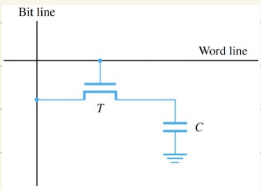
Memory Organization



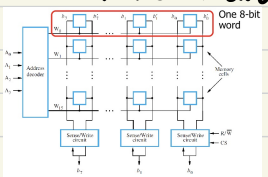
$A_0 - A_3$: 4 bits
$w_0 - w_{15}$: $16 = 2^4$
$b_0 - b_7$: 8
$\Rightarrow 16 \times 8$ memory

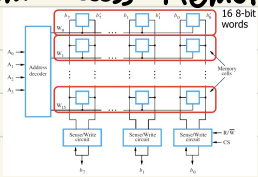


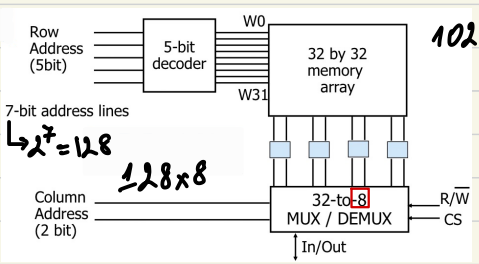
SRAM → Static Random Access Memory



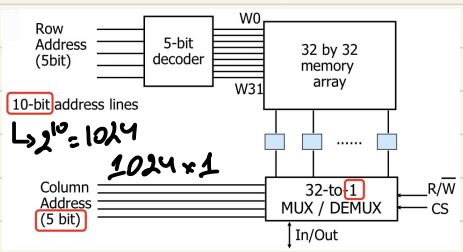
DRAM → Dynamic Random Access Memory







1024-bit



128x8 organization	1024x1 organization
• 7 address pins	• 10 address pins
• 8 data pins	• 1 data pin
• 2 R/W + CS	• 2 R/W + CS
• 2 power + ground	• 2 power + ground
---- +	---- +
19 pins	15 pins

k x n organization
$\log_2 k$ address pins
n data pins
+4 constant pins

Quiz 12.2 32x16
5 address pins
16 data pins
4 constant pins
25 pins

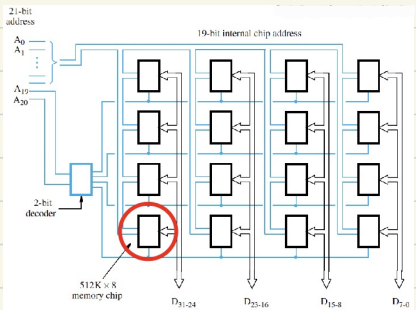
512 = 16 x 32 32 bit word
MDR = 32 MAR = $\log_2 32 + 1 = 6$

Memory address lines vs. byte addressability

? x 32 memory organization

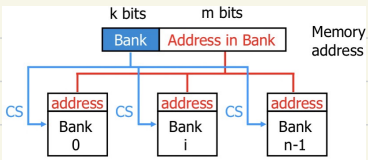
Memory reads/writes 32-bit word lines at once

Programs address memory by byte

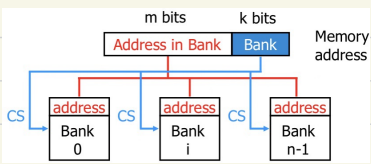


Interleaved Memory

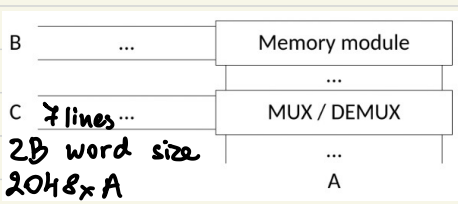
consecutive addresses
↳ same memory



consecutive addresses
↳ consecutive banks



Self-Study 7



$\log_2 2048 = 11$ address lines

$C = 7$ lines

$\Rightarrow B = 4$ lines

$2B = 16$ b $\Rightarrow A = 16$ data lines

$\Rightarrow 2048 \times 16$

(a) 128-bit words

$\log_2 (2^{21} \times 2^{-7}) = 21$

(b) 64-bit words

$\log_2 (2^{28} \times 2^{-6}) = 22$

(c) 32-bit words

$\log_2 (2^{28} \times 2^{-5}) = 23$

(d) 16-bit words

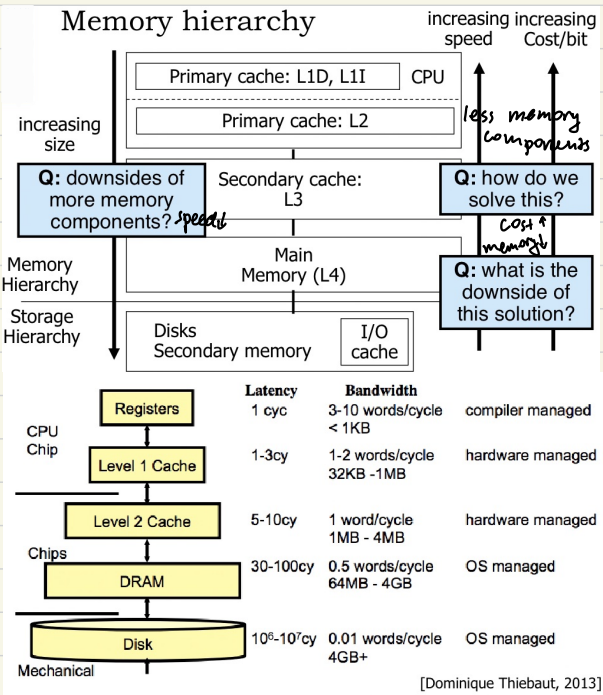
$\log_2 (2^{28} \times 2^{-4}) = 24$

$32\ \text{MiB} = 2^{25}\text{B} = 2^{28}\text{b}$

32 MiB word addressable memory chip

address lines per word size

Lecture 13 – Caching



READ operation:
- not in cache (MISS)
  - copy block into cache
  - read out of cache
- in cache (HIT)
  - read out of cache

WRITE operation:
- not in cache (MISS)
  - write in main memory
- in cache (HIT)
  - write in cache
  - and either:
    - → write in main memory
    - → set modified (dirty) bit, and write later

main memory: read → read-through; write → store through

locality:
- spatial – close by data (array)
- temporal – used before data

L1 cache

cache hit ratio: h

access time cache: c

cache miss ratio: $1-h$

access time main memory: m

average access time: $c+(1-h)m$

parameters depend on:
- → application, input (data-driven)
- → cache policy

Quiz 13.1

speedup = time (old) / time (new)

$m = 400ns$

$c = 40ns$

$h = 60\%$ $\quad 1-h = 40\%$

$$\text{speedup} = \frac{m}{c+(1-h)m}$$

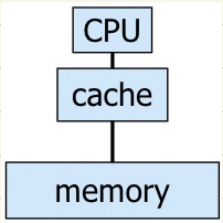
$m' = 480$

$h' = 95\%$

$$= \frac{400ns}{40ns + 40\% \cdot 400ns}$$

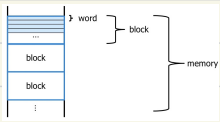
$$s = \frac{480}{40+24} = \frac{400ns}{40ns+160ns} = 2$$

$= 7.5$



Transparent to CPU (and programmer)
Storage to hold data
and administration (in/out, clean/dirty)

Block based:
→ amortize admin overhead
→ efficient lookup (hit or miss?)
→ efficient memory access



access multiple memory words at a time

- Access word w in memory block B
- Is block B in the cache?
- If not (MISS)
  - → get block B from memory
  - → calculate where to place in cache
  - → is position free?
  - → if not, evict block at B's place from the cache
  - → write B to its cache location
- Access w in the cache

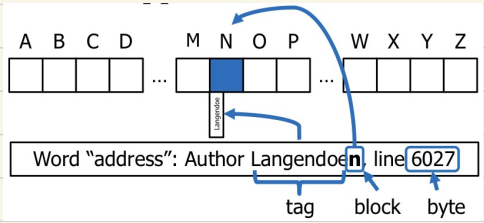
cache – smaller than memory
→ multiple memory blocks map to the same cache block

mapping can be:
→ static (direct mapped)
→ dynamic (fully associative)
→ mixed (x-way set associative)

hit ratio determined by:
→ mapping strategy
→ locality of reference

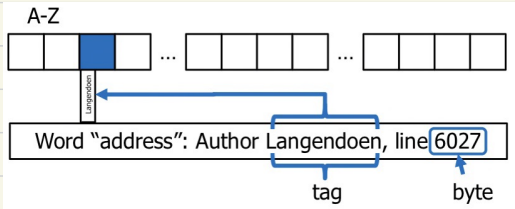
main memory → library
cache → bookcase at home
block → book
byte → line

Direct Mapped Cache



(books)
easy to look up blocks in the cache (bookcase)
not flexible – high thrashing replacing blocks in cache

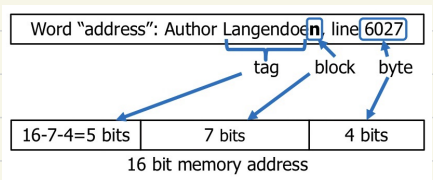
(Fully) Associative Cache

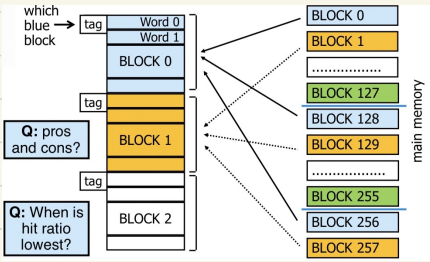


difficult to look up blocks (books) in the cache (bookcase)
check labels (in parallel)
very flexible → high hit ratio

Direct Mapped

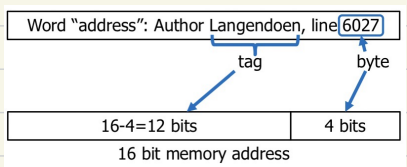
16 bit memory address
16 bytes in a block (4b)
128 blocks in cache (7b)

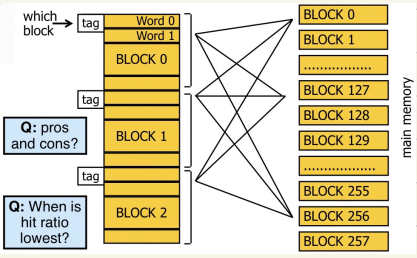




Fully Associative

16 bit memory address
16 bytes in a block (4b)
4k blocks in memory (12b)

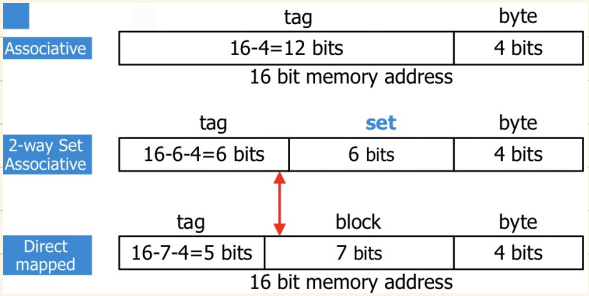


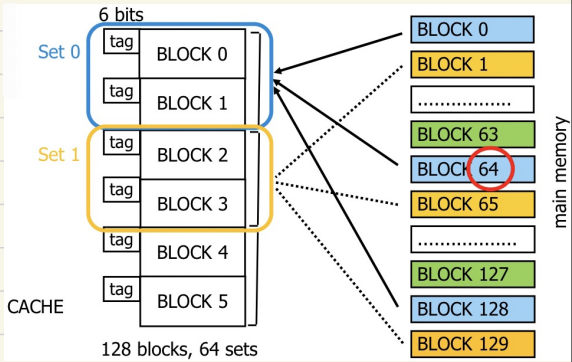


Set Associative Cache

trade-off: performance vs. flexibility
blocks into sets
- direct mapped

parallel search in set
- associative





Quiz 13.2

main memory: 3 GB
Cache: 512 blocks of 64 byte
Cache: 64-way set-associative
byte addressable (B)
3GB < 4GB = 2^{32}B → 32b

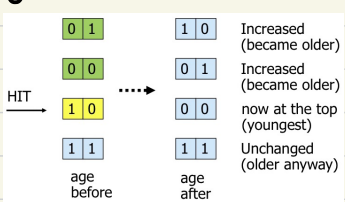
tag	set (3b)	byte (6b)

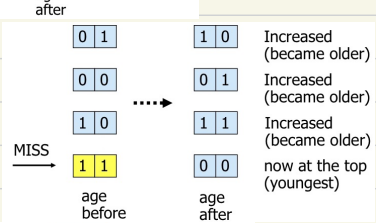
512b = 8 blocks
64 bytes
64-way tag = 32-3-6 = 23b

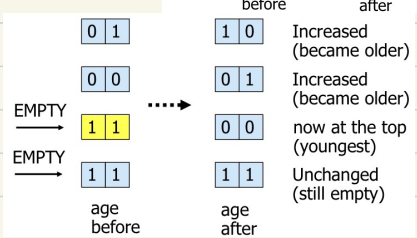
Cache Replacement Algorithms

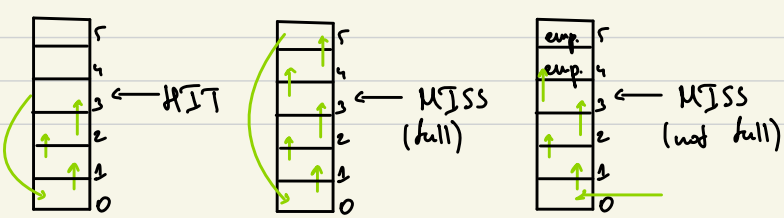
Least recently used (LRU)

- keep "time stamps" of accesses in counters
  - 2^x-way associative → x-bit age counter per block
- Basic algorithm:
  - hit:
    - → Reset hit counter
    - → Increment lower counters
  - miss, set full:
    - → Replace block with highest counter, reset this counter
    - → Increment other counters
  - miss, set not full:
    - → Place in empty spot
    - → Increment other counters



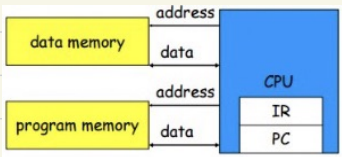






Lecture 14 – Pipelining

Harvard Architecture



Exploit difference in locality
- instructions → spatial + temporal
- data → temporal + spatial

Separate data and instruction paths

Quiz 14.1

1 laundry room, washer + dryer + folding mat
Alice, Bob, Claire, Derek ×1 load each
1 load = wash (30) + dry (40) + fold (20)

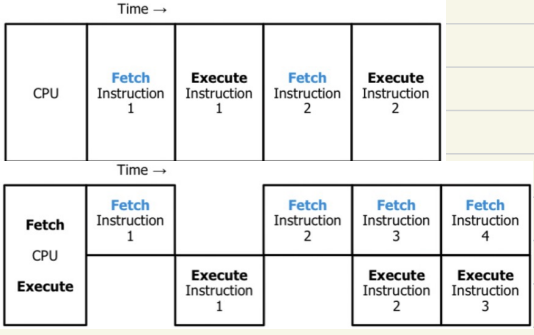
30 40 20 → 30+4×40+20 = 210
30 40 20 → 30+40+20 = 90
30 40 20
30 40 20

Performance

- Latency (L)
  - → time for 1 instr.
  - → lower is better
- Throughput (T)
  - → the # instructions per time unit
  - → higher is better
  - → time per instr. ↓ (on avg.)

Improvements

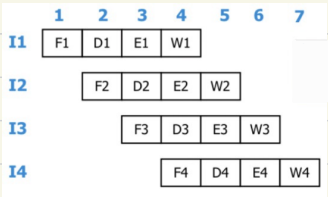
clock frequency → latency boost (power wall)
threads & cores → throughput boost (parallel programming)
pipelined execution → throughput boost (limited effect)
dedicated circuitry → latency boost (one-trick pony)

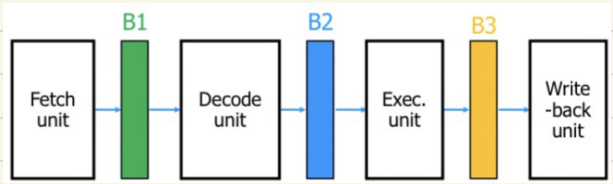


Instruction stages

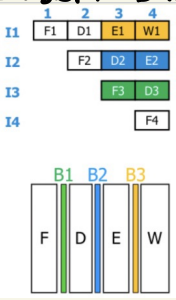
1. Fetch instruction
2. Decode instruction and fetch operands
3. Execute operation
4. Write result

simplified 4 stage model





Insert buffers to allow pipeline stages to act independently

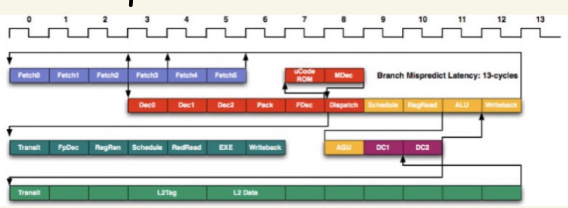


B1 (F→D):
→ instruction I3

B2 (D→E):
→ the source operand of I2
→ the operation
→ the specification of the destination operand

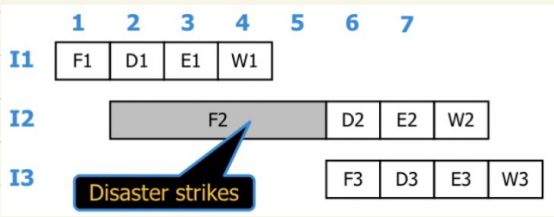
Pipeline logic:
→ data path
→ operands specifiers
→ control signals

B3 (E→W):
→ the result of the execution of I1
→ the specification of the destination operand



Harvard organization
multiple cycles per stage
extra stages (cache)
even more cycles per stage
complications: pipeline hiccups

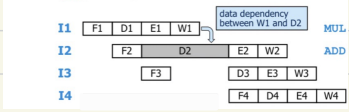
Pipeline stalls



Hazards (causes):
- cache miss [fetch, decode]
- dependency between instr. [decode] output → input
- branching [exec]
- long operation [exec] division

Data Dependency

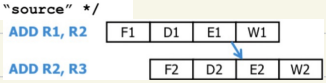
```
MUL  R2,R3,R4    /* R4 destination */
ADD  R5,R4,R6    /* R4 source */
```

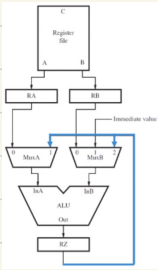



```
ADD  R1, R2     /* destination */
MUL  R3, R4
ADD  R2, R3     /* source */

MOV  R1, R2     /* destination */
ADD  $100, R2   /* source! */

MOV  4(R3),R4   /* destination */
ADD  8(R4),R5   /* "source" */
```

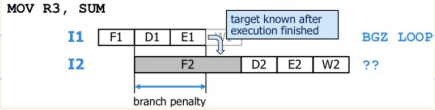




Data forwarding
Add last path from ALU output to input

Branching

```
BGZ LOOP
MOV R3, SUM
```




```
ADD R6,R7,R8
BGZ R3 LOOP
MOV R8, SUM
```

- Penalty = 1 cycle
- Use idle slot

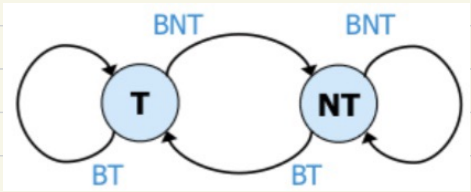
```
BGZ R3 LOOP
ADD R6,R7,R8
MOV R8, SUM
```

branch delay slot: always executed before the branch effects

Unconditional branching:
- → target known after decoding
- → branch penalty reduces to 1 cycle

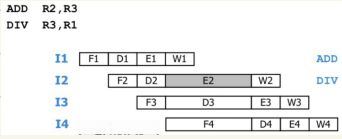
Static prediction:
- → backwards branches likely to be taken (loops)
- → forward branches mostly skipped (rule vs exception)
- → sign bit (or compiler directed)

Dynamic prediction:
- → maintain info per branching instr. (branch target buffer)
- → do as before



Long operations

```
ADD  R2,R3
DIV  R3,R1
```



Execution time of a program T

Instruction count N

cycles per instruction, CPI S (4)

Clock rate R (1GHz)

without pipelining: $T = \frac{NS}{R}$

with n-stage pipeline: $T' = \frac{T}{n}$

$S \uparrow \rightarrow$ stalls

Quiz 14.2

$h_i = 0.95$ $h_{branch} = 0.90$

$h_d = 0.90$ $p_{branch} = 1$

$p_c = 16$ cycles $f_{branch} = 0.20$

$f_d = 0.30$

cache misses

$= (m_i + f_d \times m_d) p_c$

$= (0.05 + 0.30 \times 0.10) 16$

$= 1.28$

branching

$= f_b \times m_b \times p_b$

$= 0.20 \times 0.10 \times 1$

$= 0.02$

$p_{total} = 1.28 + 0.02 = 1.3$ cycles

Self-Study 8

32 bit words

16 MiB word addressable memory

256 KiB direct mapped cache

128 bytes block size

$\frac{\text{block size}}{\text{word size}} = \frac{128\text{B}}{32\text{b}} = 32 \frac{\text{words}}{\text{block}}$

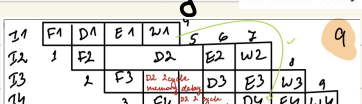
$\frac{\text{cache size}}{\text{block size}} = \frac{256\text{ KiB}}{128\text{ B}} = 2048 \frac{\text{blocks}}{\text{cache}}$

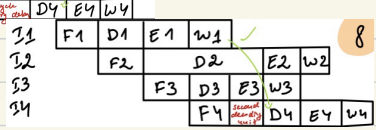
32 b word $\Rightarrow \log_2 32 = 5$-bit addressing

2048 blocks $\Rightarrow \log_2 2048 = 11$-bit addressing

```
1 Add R1, R2, R3
2 Move (R4), R5
3 Move #10, R6
4 Add R3, R4, R7
```

$p_{memory} = 2$ cycles





8-way set associative

512 blocks of 128 bytes

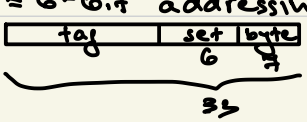
8 GiB memory

byte addressable

128 byte-addressable blocks $\Rightarrow \log_2 128 = 7$-bit addressing

$\frac{512}{8} = 64$ sets $\Rightarrow \log_2 64 = 6$-bit addressing

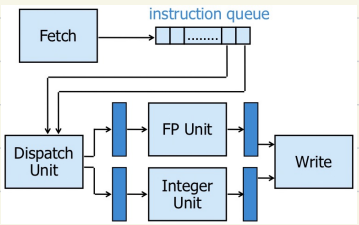
8 GiB $= 2^{33}$ B $\rightarrow$ 33 bits total

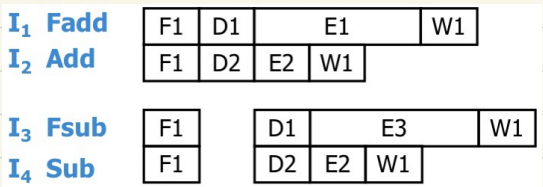


$33 - 6 - 7 = 20$ bits tag

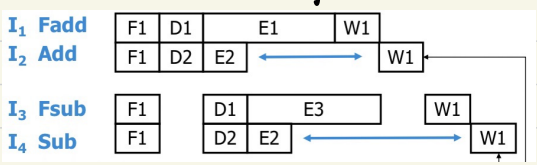
Lecture 15 – Parallelization and Virtual Memory

Superscalar Execution





in-order issuing of instructions
out-of-order completion



in-order issuing of instructions
program-order completion

Amdahl's Law

Parallelization:
- → throughput ↑
- → latency =
- → program runtime T↓

Cons:
- → Parallel programming is difficult
- → Not all code can be parallelized:
  - data dependencies
  - sequential algorithms

$$T_p = T_s \left(f_s + \frac{f_p}{p} \right)$$

$$\text{speedup} = \frac{1}{f_s + \frac{f_p}{p}}$$

f_p – fraction of code that can be parallelized
$1 - f_p = f_s$ – sequentially executed code
T_s – time to execute program sequentially
T_p – time to execute program using p processors

Quiz 15.1

$f_p = 90\%$ $f_s = 10\%$

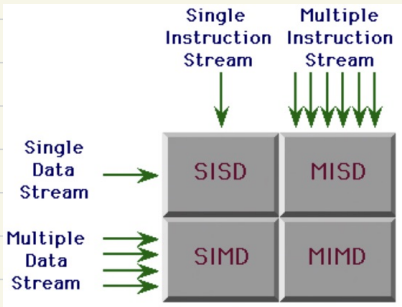
a) $p = 3$, $s = ?$ b) $s_{max} = ?$ c) $s = 5$ $p = ?$

a) $s = \frac{1}{10\% + \frac{90\%}{3}} = \frac{1}{40\%} = 2.5$

b) $p = \infty$ $s_{max} = \frac{1}{10\%} = 10$

c) $5 = \frac{1}{10\% + \frac{90\%}{p}} = \frac{1}{20\%}$ → $0.2 = 10\% + \frac{90\%}{p}$ → $\Rightarrow p = 9$

Flynn's Taxonomy



SISD → Single Instruction, Single Data
→ conventional system

SIMD → Single Instruction, Multiple Data
→ one instruction on multiple data streams

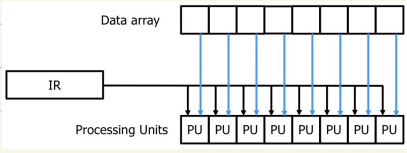
MIMD → Multiple Instruction, Multiple Data
→ multiple instruction streams on multiple data streams

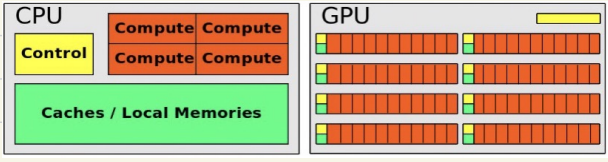
MISD → Multiple Instruction, Single Data

SIMD

CPU: A lot of cache and control logic + reasonable compute logic

GPU: Small cache and control logic + a lot of compute logic

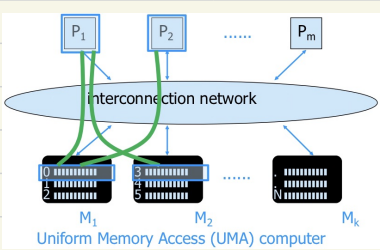




Shared-memory Multiprocessors

Uniform Memory Access (UMA) architecture

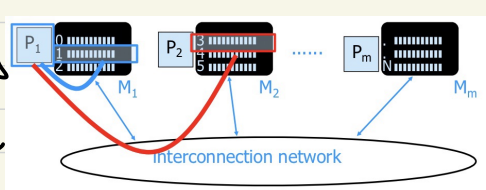
Any processor can access (transparently) directly any memory



Non-Uniform Memory Access (NUMA)

Any processor can access directly any memory, but not at the same speed.

realization in hardware or in software (distributed shared memory)

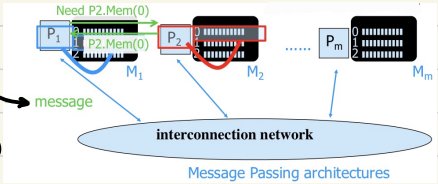


Distributed memory architecture

but not shared

not transparent

Any processor can access any memory, but sometimes through another processor (messages)



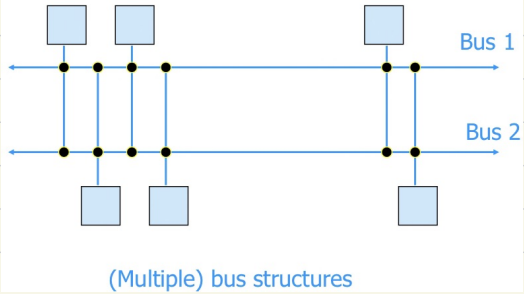
Message Passing architectures

Interconnection networks

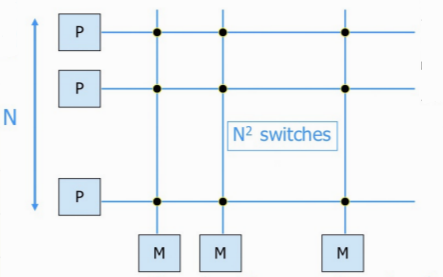
Difficulty in building systems with many processors – interconnections

Parameters:

- Diameter – maximal distance between any two processors
- Degree – maximal number of connections per processor
- Total number of connections = cost
- Bisection width – largest number of simultaneous messages



(Multiple) bus structures



Cross-bar interconnection network

Hypercubes

Non-uniform delay – NUMA architecture

$n \times 2^{n-1}$ connections

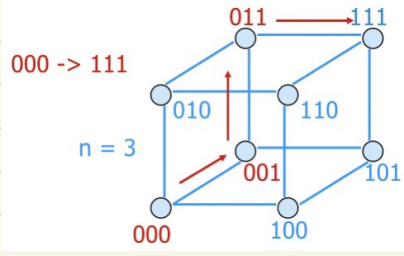
Connected processors differ by 1 bit

Routing:

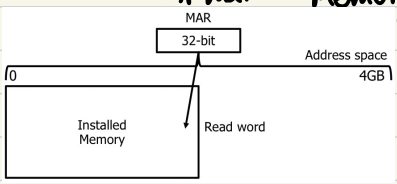
→ scan bits from right to left

→ if different, send to neighbour with same bit different

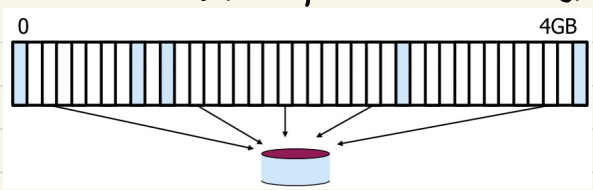
→ repeat until end



Virtual Memory



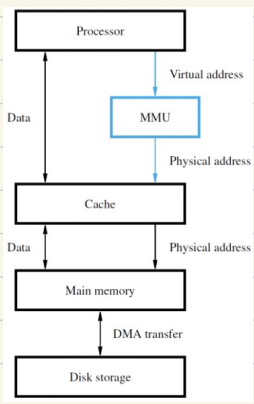
Divide address space in chunks
Store more chunks on disk
When chunk is needed, move to main memory



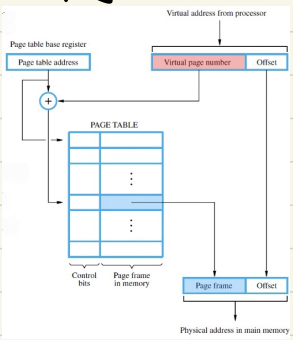
Caching	Virtual Memory
Blocks	Pages
Cache miss	Page fault
Tag field	Page number
Byte/word field	Page offset
Mapping function	Page table

Memory Management Unit
- → Hardware support
- → mapping of pages in memory
- → access control

Operating System
- → sets up MMU and paging tables
- → handles misses

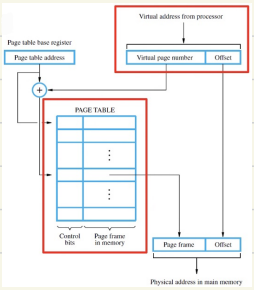


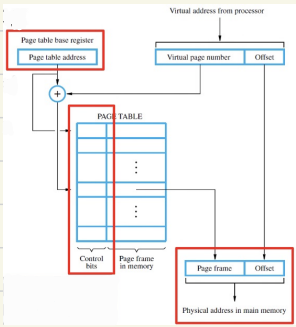
Demand paging
- From virtual address
- through page table
- to physical address



Logical address
- covers complete address space

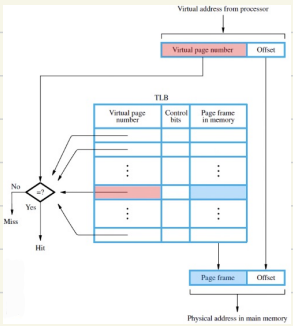
Page table
- stored in main memory (OS)
- translates virtual page number to a physical page frame





CPU contains special register that stores the starting location of the page table (one per process). Combining page frame and offset gives the complete physical memory address.

Large address space → large page table
Page table does not fit in cache



Translation Lookaside Buffer (TLB)

- Special cache for small part of the page table
- Look up virtual page number to find page frame

- If part of page table not found in TLB → regular cache miss
  If page frame found in page table, but marked as invalid → page fault
- If page frame not mapped in page tables → segfault

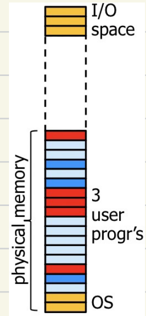
- MMU issues a trap (interrupt caused by execution)
- CPU
  - → aborts the instruction
  - → switches to kernel mode
- OS: exceptional handler
  - → switches for a free page frame
  - → loads the requested page from disk into memory
  - → updates the page table
  - → restarts the aborted instruction

Quiz 15.2

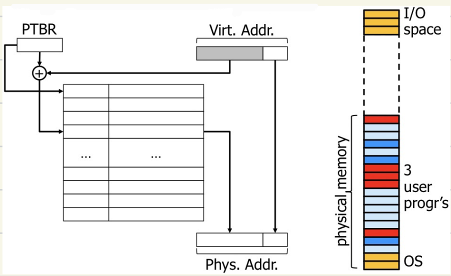
42-bit PC
word size = 16b
word addr. RAM = 8 GiB
OS memory = 32 GiB
pages = 4 KiB
32 MiB

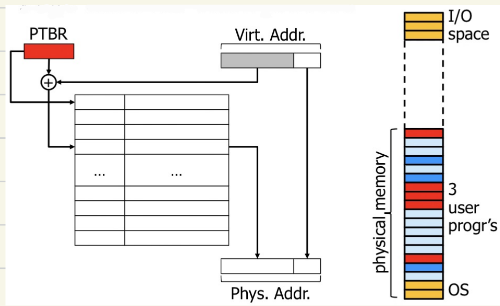
4 KiB offset = 2^{12} B → 12b
32 GiB = 2^{35} B → 35b
35 − 12 = 23b virtual page number
8 GiB = 2^{33} B → 33b
32b = 4B
33b − 12b = 21b
27b = 6b + 21b
control bits = 6b

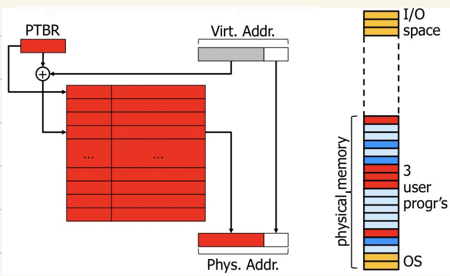
Prevent programs from accessing
- OS internals (page tables, I/O devices)
- each others memory

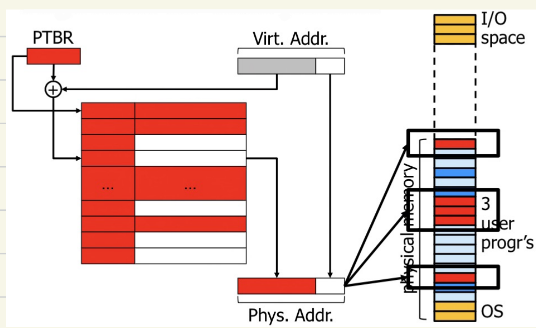


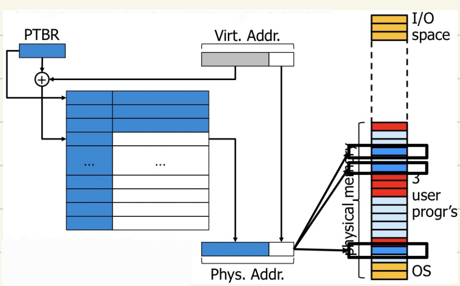
OS loads correct page table for each process
- OS maps only what is allowed
- Other parts of the memory are unreachable

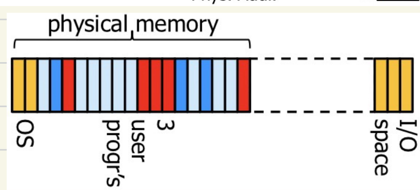












Separate 'kernel' mode for OS processes
- Give access to special instructions:
  → access 'page-table base register', ...

Self-Study 9

Amdahl's Law

① $T_s = 32s$, $T_p < 16s$, $f_p = 87\%$, $f_s = 13\%$, $p = ?$

$$s = \frac{T_s}{T_p} > 2 \qquad \frac{1}{f_s + \frac{f_p}{p}} \geq 2 \qquad f_s + \frac{f_p}{p} \leq \frac{1}{2}$$

$$p \geq \frac{0.87}{0.37} \sim 2.4 \qquad \frac{0.87}{p} \leq 0.37 \qquad 0.13 + \frac{0.87}{p} \leq 0.5$$

$\Rightarrow$ at least 3

② $T_s = 42s$, $p = 252$, $T_p = 28s$, $f_p = ?$, $f_s = 1 - f_p$

$$T_p = T_s\left(f_s + \frac{f_p}{p}\right) \qquad \frac{2}{3} = 1 - \frac{251}{252} f_p \qquad f_p = \frac{84}{251}$$

$$\frac{28}{42} = 1 - f_p + \frac{f_p}{252} \qquad f_p = -\frac{1}{\frac{2}{3}} \cdot -\frac{252}{251} \qquad f_p \sim 0.33$$

Virtual Memory

① page = 24 KiB = 3×2^{13} B

42×2^{20} entries = 21×2^{21}

total addressable virtual memory = $3 \times 2^{13} \times 21 \times 2^{21}$ B = 63×2^{24} B = 63×16 GiB = 1008 GiB $\sim$ 1 TB/TiB

② 64 GiB memory $\rightarrow 2^{36}$ B

16 GiB page table $\rightarrow 2^{34}$ B

24-bit physical page number, 6 control bits, 2 KiB page $\rightarrow 2^{11}$ B

30-bit entry $\rightarrow$ 32 bits

$$\frac{2^{34}\text{ B}}{32\text{ b}} = \frac{2^{34}\text{ B}}{2^2\text{ B}} = 2^{32} \text{ pages}$$

$$2^{32} \cdot 2^{11}\text{ B} = 2^{43}\text{ B} = 8\text{ TiB}$$

total addressable virtual memory