Introduction to
Information Science and Engineering

Step1. Basic structure of commercial computers and programming

http://archlab.naist.jp/Lectures/ARCH/ca01_0301_0303_0702/ca010307e.pdf

Download the template and submit through UNIPA.

http://archlab.naist.jp/Lectures/ARCH/ca01_0301_0303_0702/ca010307e.docx

in http://archlab.naist.jp/Lectures

Outline of this class

1. **Basic of Von Neumann computers**
   - It seems good way to start from CPU, but the power efficiency is lowest.
   - No evolution in Von Neumann computers due to physical limitations

2. **Power efficient Non-Von Neumann computers**
   - Programmability / accuracy are sacrificed but high-speed and low-power.
   - You'd better learn Non-Von Neumann computers for zero-emission soon.

3.4. Operating system	… to make computers easy to use
5.6. Software	… to translate algorithm to hardware
7.8. Algorithm	… to solve specific problems

CA01:Basic structure of computers and programming

Goal of today

Q1. Semiconductors are imperfect switches. What is the mechanism for making complete switches?
半導体は不完全なスイッチである.完全なスイッチにできる仕組みは?

Q2. What are three main components of a computer.
コンピュータの主要構成要素を3つ挙げよ

Q3. What is the background of CISC in the past, and RISC now?
昔はCISC,今はRISCを指向する背景は?

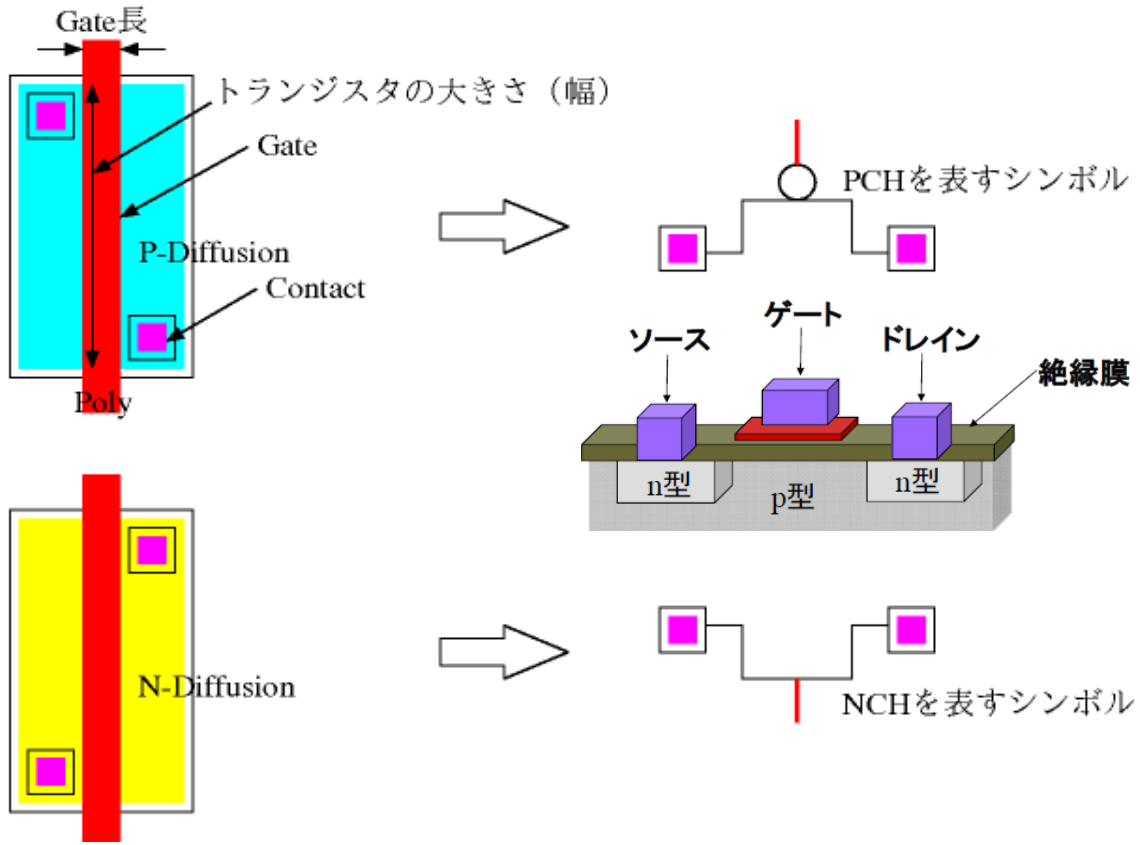
Q4. In network, which order is correct? Big endian or little endian? Which order in Intel and ARM?
ビッグエンディアンとリトルエンディアン,ネットワークに流して良いのはどっち? Intel,ARMはどっち?

Q5. getname () {char buf [80]; gets (buf); ...} How is this program attacked?
getname() {char buf[80]; gets(buf);...} このプログラムが乗っ取られる仕組みは?

Q6. Your program is correct. But 100 times slower than colleague's program. What should you do?
あなたのプログラムは無駄がなく結果も正しい.でも同僚のプログラムより100倍遅い.気付くべき点は?

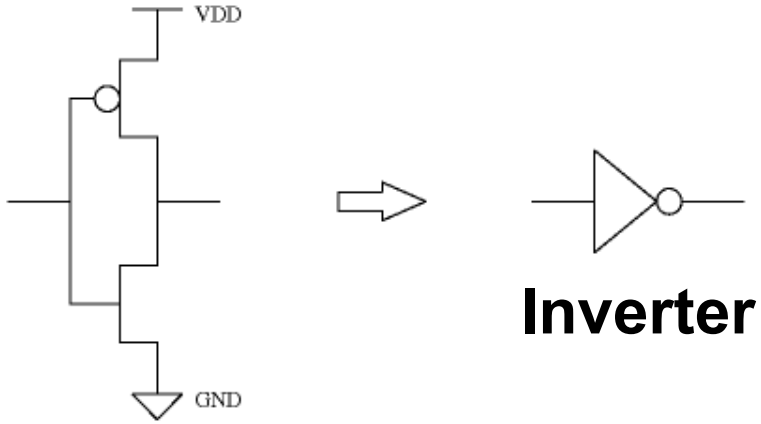
FPU has huge number of transistors

Computers are made of imperfect switches
pMOS transistors pass 1 w/ gate=0
nMOS transistors pass 0 w/ gate=1



Logic elements with pMOS and nMOS

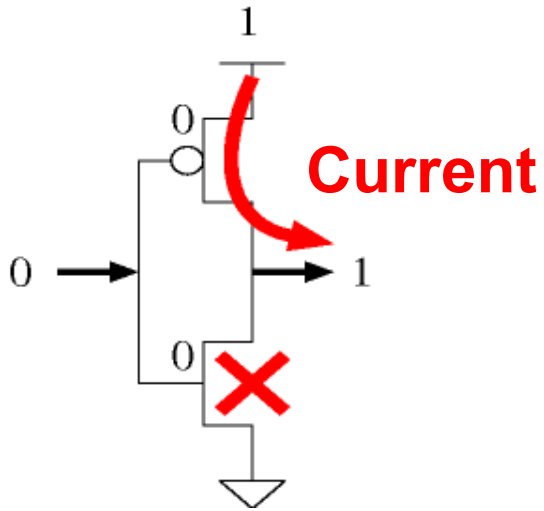
Complementary MOS gate

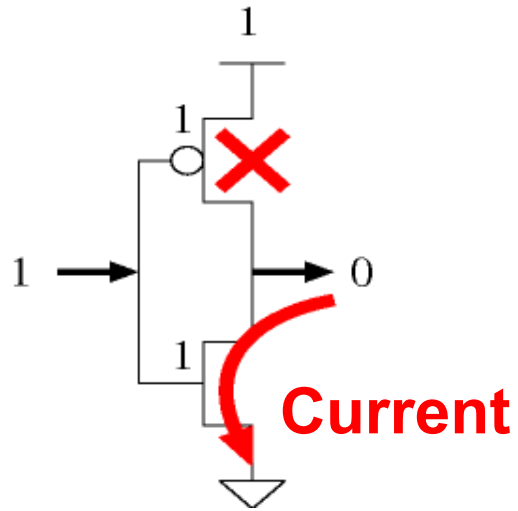


Inverter

PCH is on w/ in=0 ⇒ pass 1

NCH is on w/ in=1 ⇒ pass 0





Logic elements with pMOS and nMOS

Calculation needs 2in-1out logical switches

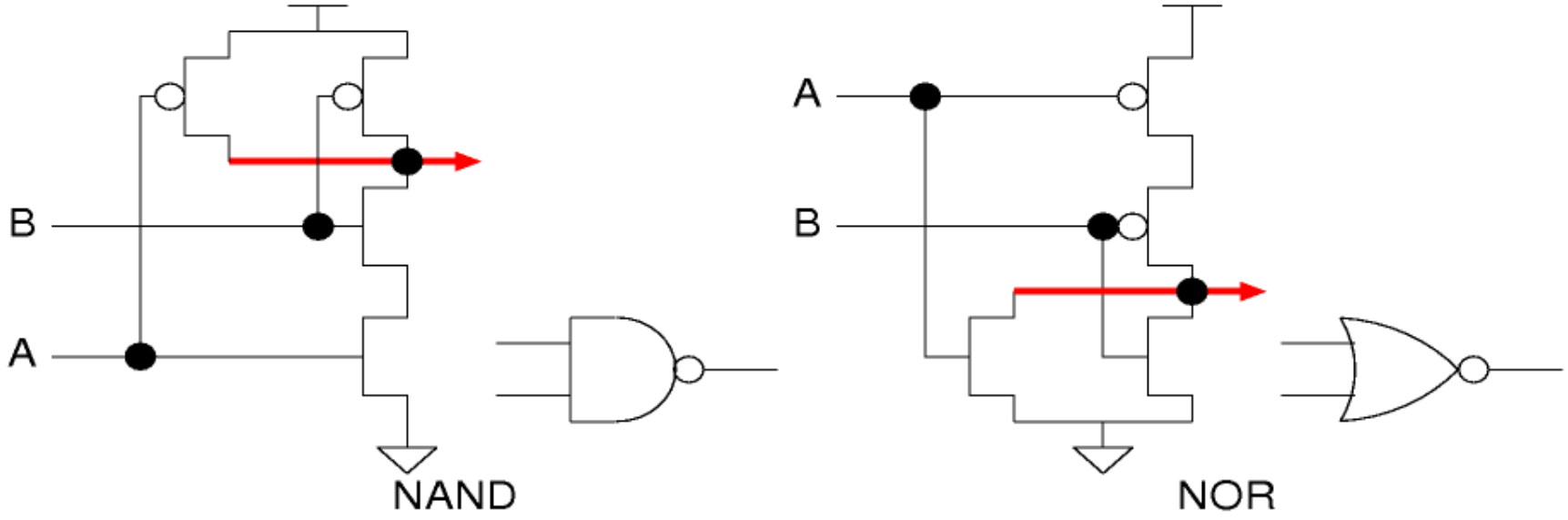
AND … if all-in=1 out=1 else out=0

OR … if all-in=0 out=0 else out=1

Minimum hardware can implement

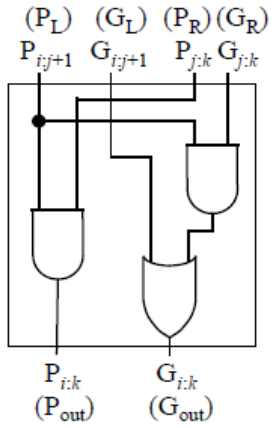
NAND … if all-in=1 out=0 else out=1

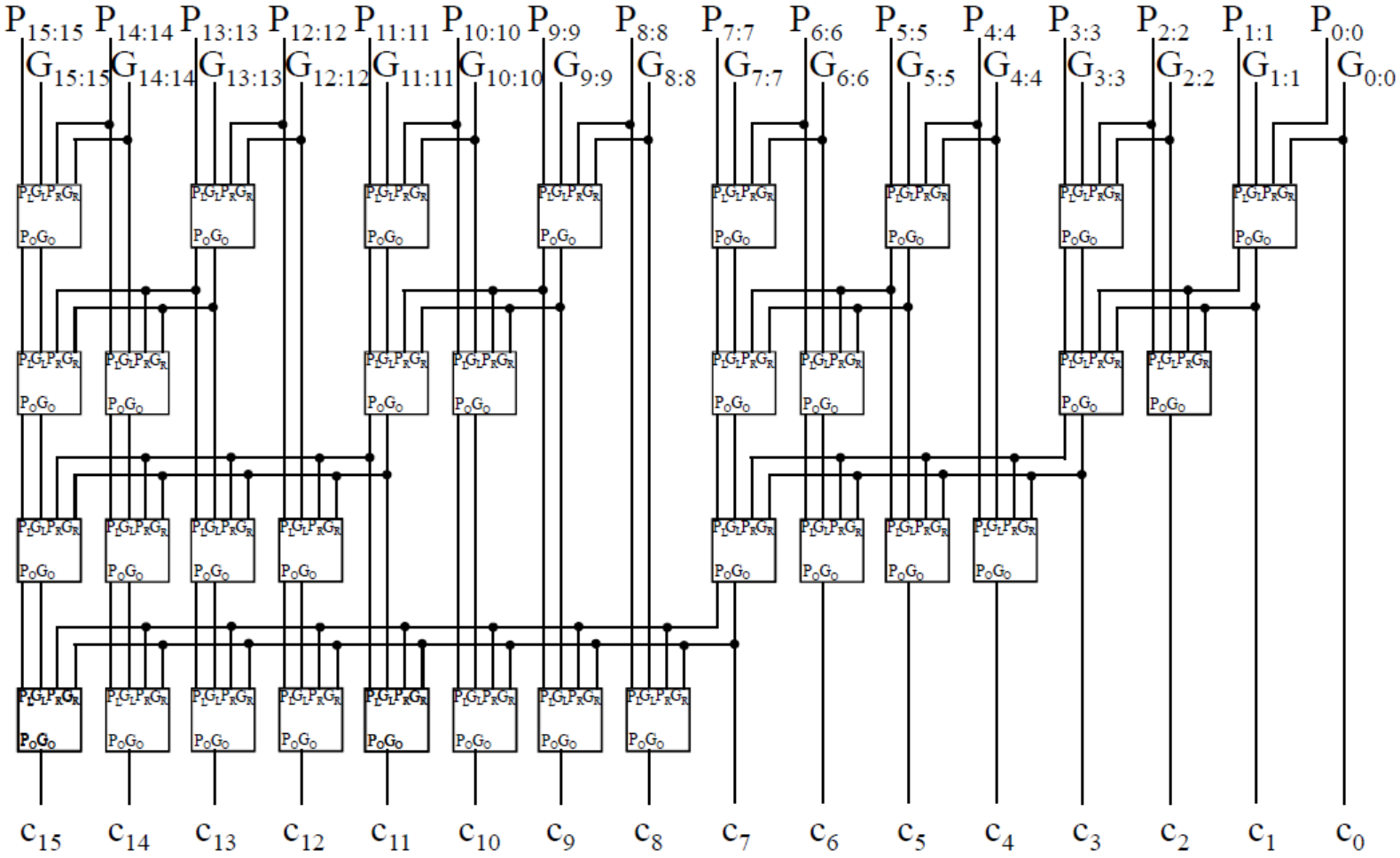
NOR … if all-in=0 out=1 else out=0

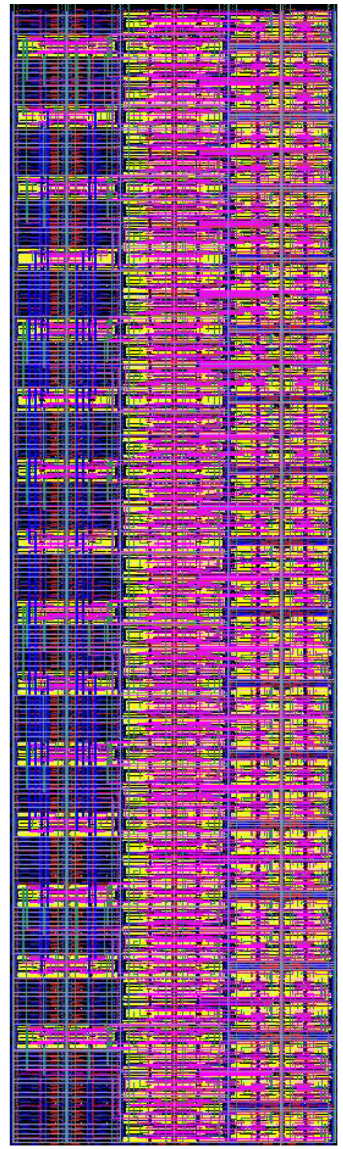


Arithmetic unit with logical elements

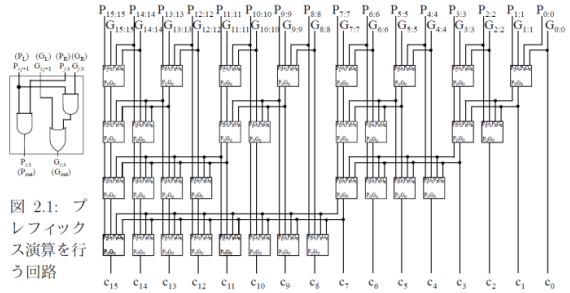
N-bit arithmetic requires Log N stages



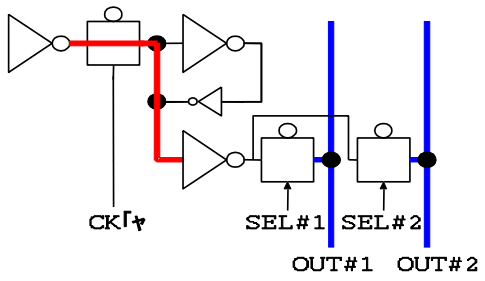




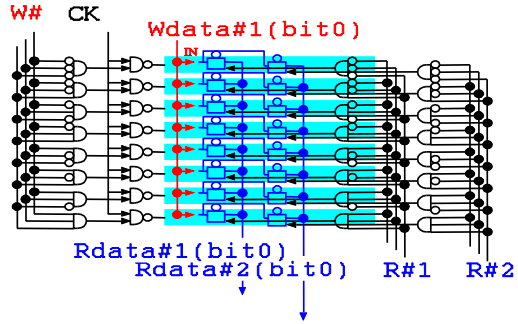
Basic units with Logic elements



ALU

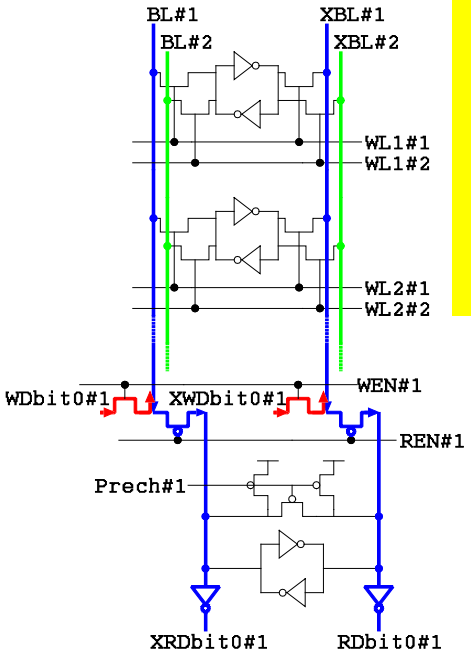


Flip Flop

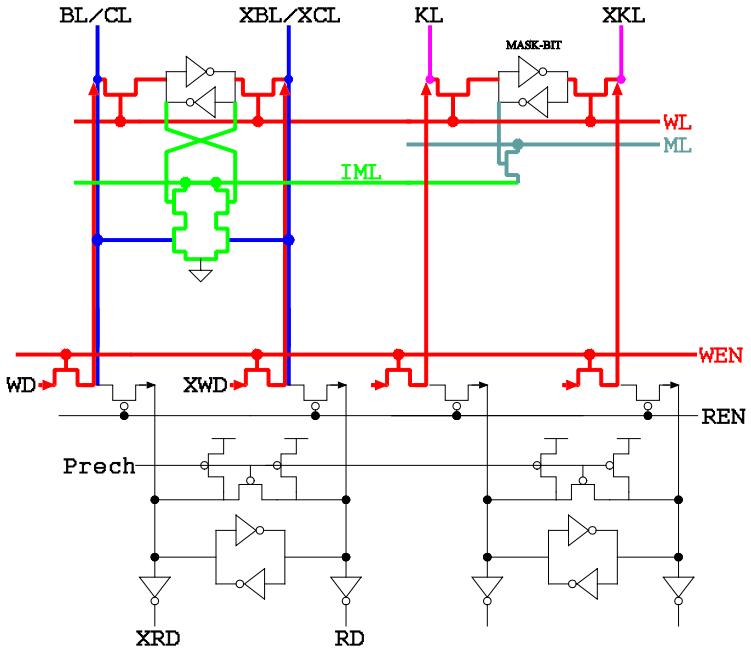


Register File

RAM (Random Access Memory)



CAM (Content Addressable Memory)



Goal of today

Q1. Semiconductors are imperfect switches. What is the mechanism for making complete switches?
半導体は不完全なスイッチである.完全なスイッチにできる仕組みは?

Q2. What are the three main components of a computer.
コンピュータの主要構成要素を3つ挙げよ

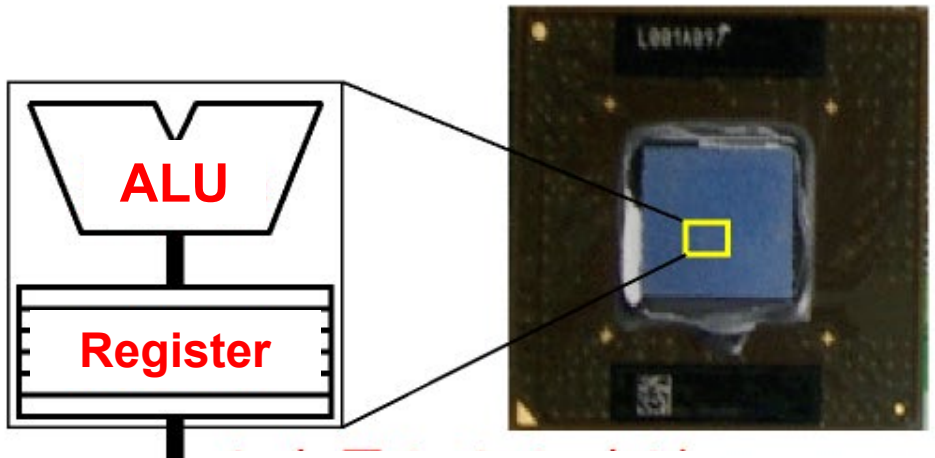
Q3. What is the background of CISC in the past, and RISC now?
昔はCISC,今はRISCを指向する背景は?

Q4. In network, which order is correct? Big endian or little endian? Which order in Intel and ARM?
ビッグエンディアンとリトルエンディアン,ネットワークに流して良いのはどっち? Intel,ARMはどっち?

Q5. getname () {char buf [80]; gets (buf); ...} How is this program attacked?
getname() {char buf[80]; gets(buf);...} このプログラムが乗っ取られる仕組みは?

Q6. Your program is correct. But 100 times slower than colleague's program. What should you do?
あなたのプログラムは無駄がなく結果も正しい.でも同僚のプログラムより100倍遅い.気付くべき点は?

Computer with Basic units



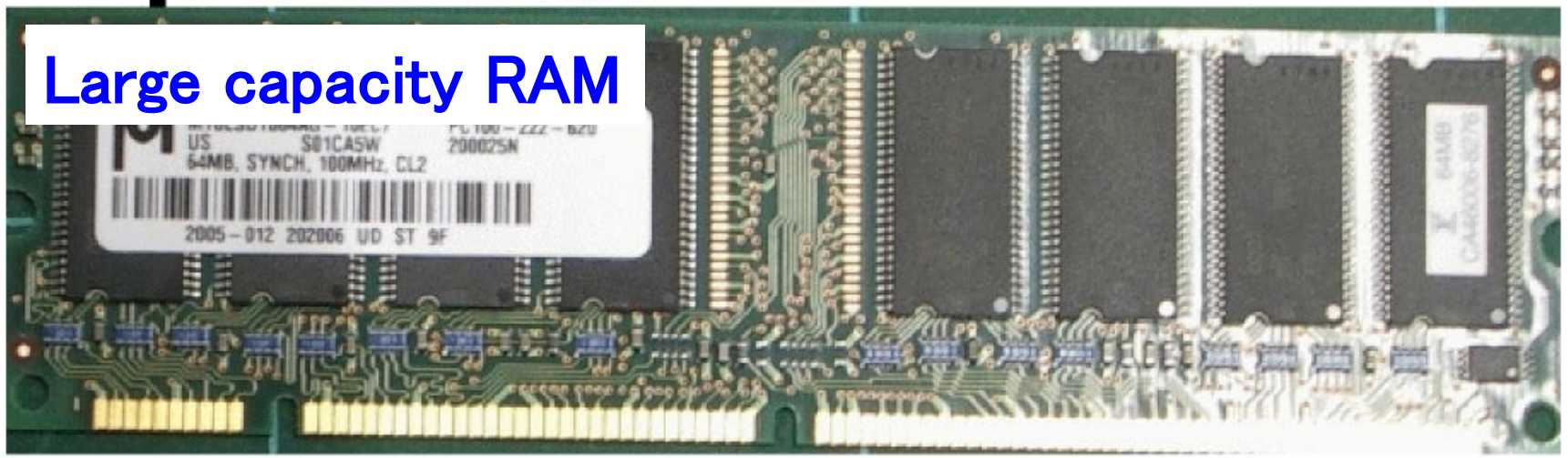
CPU（Central Processing Unit）

I/O device

Main Memory Unit

Small but Fast

Large but Slow



Computer = calculator with program

1. **Sample C-source**

```
% vi sample.c
main()
{
  printf("Hello.\n");
}
```

2. **C Compiler generates executables**

- **Preprocessor, Compiler, Assembler and Linker are called**
- sample.c ⇒ cxxx.i ⇒ cxxx.s ⇒ cxxx.o ⇒ sample

```
% gcc -v sample.c -o sample
…/cpp …    sample.c      …/cxxx.i
…/cc1 …    …/cxxx.i -o   …/cxxx.s
…/as  …    …/cxxx.s -o   …/cxxx.o
…/ld  …    …/cxxx.o -o   sample
```

3. **Execution**

```
% ./sample
Hello.
```

Assembly language depends on CPU architecture

Assembly source to object

- Pentium

```
main: pushl %ebp
      movl  %esp,%ebp
      pushl $.LC0
      call  printf
      leave
      ret
.LC0: .ascii "Hello.\12\0"
```

- SPARC

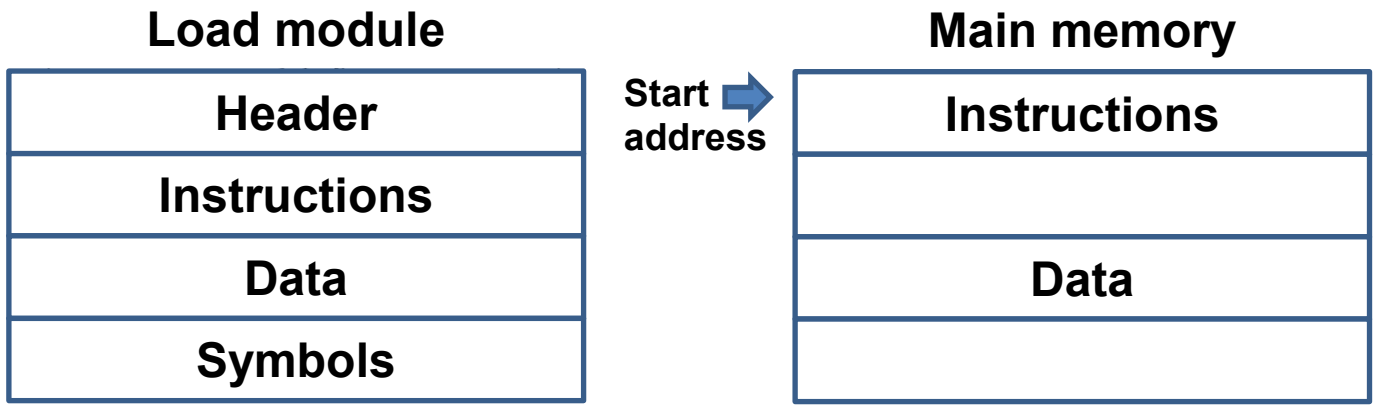
```
main: save  %sp,-112,%sp
      sethi %hi(.LLC0),%o0
      call  printf,0
      or    %o0,%lo(.LLC0),%o0
      ret
      restore
.LLC0:.asciz  "Hello\n"
```

- HPPA

```
main  ldil  LR'L$C0000,%r26
      bl    printf,%r2
      ldo   RR'L$C0000(%r26),%r26
      ldw   -148(0,%r30),%r2
      bv    0(%r2)
      ldo   -128(%r30),%r30
L$C0000 .STRING "Hello.\x0a\x00"
```

How program works on computers

Execution of load modules



CPU

- Register A
- Register B
- Register C
- Register D

Main memory

Address (Hex)	Instructions and Data
1000 1004 1008 100C 1010 1014 1018 101C 1020 1024 1028	LOAD 2000 → A LOAD 2054 → B COMP A, B BLE 1XXX STORE B → 2000 STORE A → 2054 LOAD 2004 → C LOAD 2058 → D STORE D → 2004 STORE C → 2058 …
2000 2004 2054 2058 20A8 20AC 20FC 2100 2150 2154	2345228 Hokkaido 9234646 Aomori 1327822 Iwate 6886096 Miyagi 1143411 Akita

Common Instructions

Memory instructions

Load … from memory to register
Store … from register to memory

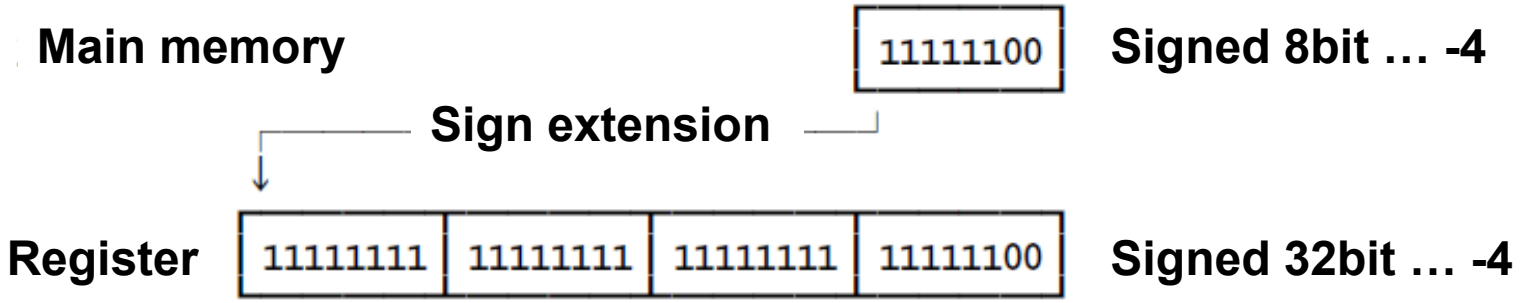
ALU instructions

Arithmetic operations	… add/sub/mult/div
Logical operations	… and/or/xor/not
Shift right arithmetic	… sra (signed div by 2)
Shift right/left logical	… srl/sll (div/mult by 2)

Control instructions

Conditional branch	… branch to specified addr
Call subroutine	… save next and branch
Return from subroutine	… restore saved addr

Common Instructions



Sign extension or zero extension

Typical Instructions

(a)	8bit load/store	Load signed byte, Load unsigned byte, Store byte
	16bit load/store	Load signed half, Load unsigned half, Store half
	32bit load/store	Load signed word, Load unsigned word, Store word
	64bit load/store	Load extended, Store extended
(b)	Arithmetic operations	Add, Subtract, Multiply, Divide, Modulo
(c)	Logical operations	Not, And, Or, Exclusive-Or
(d)	Signed shift	Shift right arithmetic
	Unsigned shift	Shift right logical
	Unsigned shift	Shift left logical
(e)	Branch	Branch, Branch on condition
	Subroutines	Call, Return

Source code to instructions

Source code

```
int R1[100];                    /* reg r1: top address of array R1[] */
int R2;                         /* R2: index for array R1[] */
R1[1] = R1[1]+8;                /* add 8 to second element of R1[] */
R1[R2]= R1[1]-R1[R2];           /* reg r2:  R2 */
```

Instructions

```
ld  r1,4,r4          r4 ← mem[r1+4]
add r4,8,r8          r8 ← r4 + 8
st  r1,4,r8          Mem[r1+4] ← r8
ld  r1,r2<<2,r5      r5 ← mem[r1+r2*4]
sub r8,r5,r9         r9 ← r8 – r5
st  r1,r2<<2,r9      Mem[r1+r2*4] ← r9
```

Computer repeats Decode, Read, Execution and Write-back.

Goal of today

Q1. Semiconductors are imperfect switches. What is the mechanism for making complete switches?
半導体は不完全なスイッチである.完全なスイッチにできる仕組みは?

Q2. What are three main components of a computer.
コンピュータの主要構成要素を3つ挙げよ

Q3. What is the background of CISC in the past, and RISC now?
昔はCISC,今はRISCを指向する背景は?

Q4. In network, which order is correct? Big endian or little endian? Which order in Intel and ARM?
ビッグエンディアンとリトルエンディアン,ネットワークに流して良いのはどっち? Intel,ARMはどっち?

Q5. getname () {char buf [80]; gets (buf); ...} How is this program attacked?
getname() {char buf[80]; gets(buf);...} このプログラムが乗っ取られる仕組みは?

Q6. Your program is correct. But 100 times slower than colleague's program. What should you do?
あなたのプログラムは無駄がなく結果も正しい.でも同僚のプログラムより100倍遅い.気付くべき点は?

RISC(Reduced Instr. Set Com.) vs CISC(Complexed …)

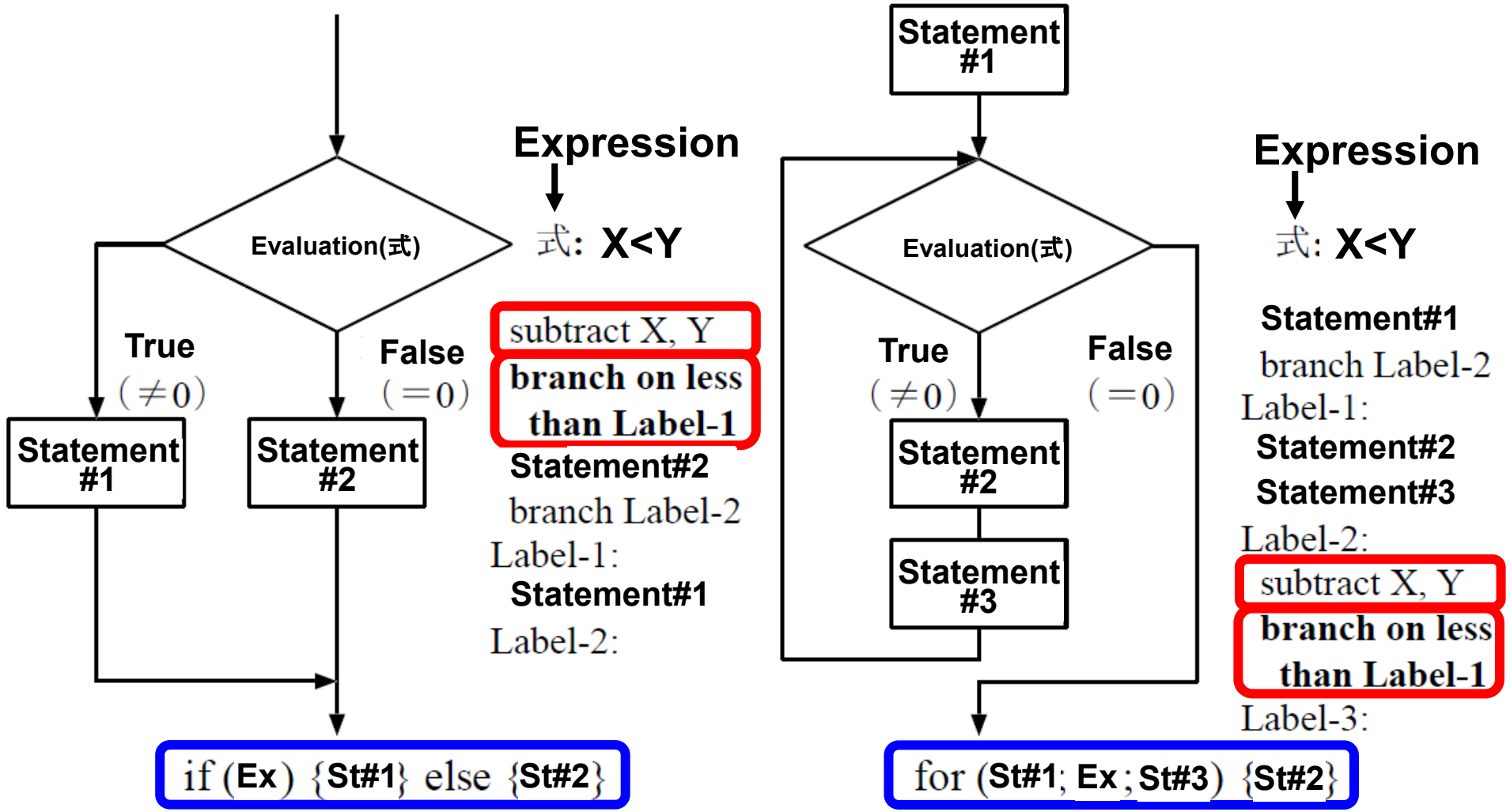
Source code and instructions

Source code	RISC Instructions
1. $int\ R1[100];$	1. $load\ (r1+4), r4$
2. $int\ R2;$	2. $add\ r4, 8, r8$
3. $R1[1] = R1[1] + 8;$	3. $store\ (r1+4), r8$
4. $R1[R2] = R1[1] - R1[R2];$	4. $load\ (r1 + (r2 \ll 2)), r5$
	5. $subtract\ r8, r5, r9$
	6. $store\ (r1 + (r2 \ll 2)), r9$

In case of CISC

Source code	CISC Instructions
1. $int\ R1[100];$	1. $add\ (r1+4), 8, r8$
2. $int\ R2;$	2. $store\ (r1+4), r8$
3. $R1[1] = R1[1] + 8;$	3. $subtract\ r8, (r1 + (r2 \ll 2)), r9$
4. $R1[R2] = R1[1] - R1[R2];$	4. $store\ (r1 + (r2 \ll 2)), r9$

Structure of program with conditional branch



Structure of “If-else” and “for” loop

Goal of today

Q1. Semiconductors are imperfect switches. What is the mechanism for making complete switches?
半導体は不完全なスイッチである.完全なスイッチにできる仕組みは?

Q2. What are three main components of a computer.
コンピュータの主要構成要素を3つ挙げよ

Q3. What is the background of CISC in the past, and RISC now?
昔はCISC,今はRISCを指向する背景は?

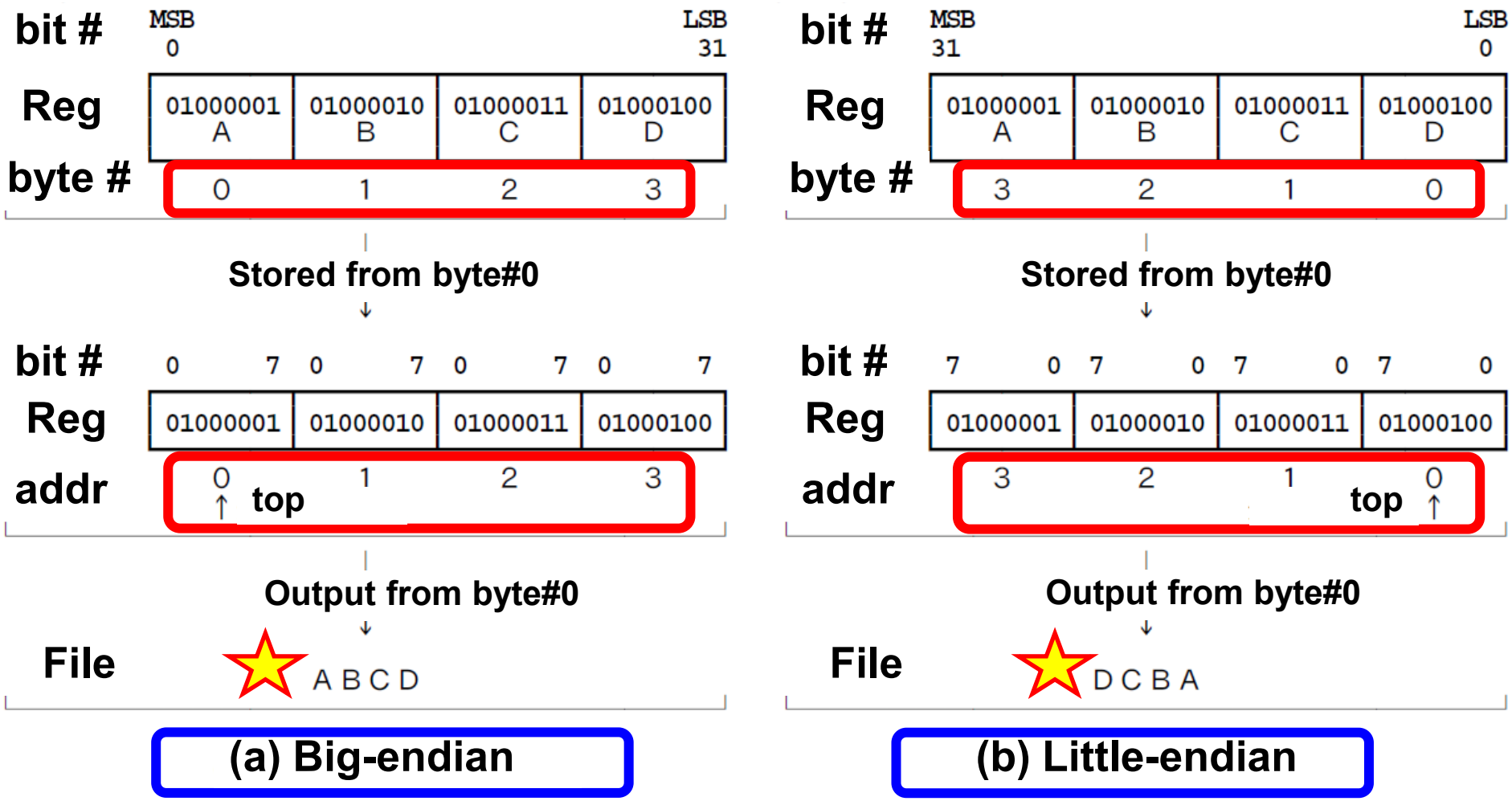
Q4. In network, which order is correct? Big endian or little endian? Which order in Intel and ARM?
ビッグエンディアンとリトルエンディアン,ネットワークに流して良いのはどっち? Intel,ARMはどっち?

Q5. getname () {char buf [80]; gets (buf); ...} How is this program attacked?
getname() {char buf[80]; gets(buf);...} このプログラムが乗っ取られる仕組みは?

Q6. Your program is correct. But 100 times slower than colleague's program. What should you do?
あなたのプログラムは無駄がなく結果も正しい.でも同僚のプログラムより100倍遅い.気付くべき点は?

Big-endian and Little-endian

Computers with different endian have no data compatibility each other

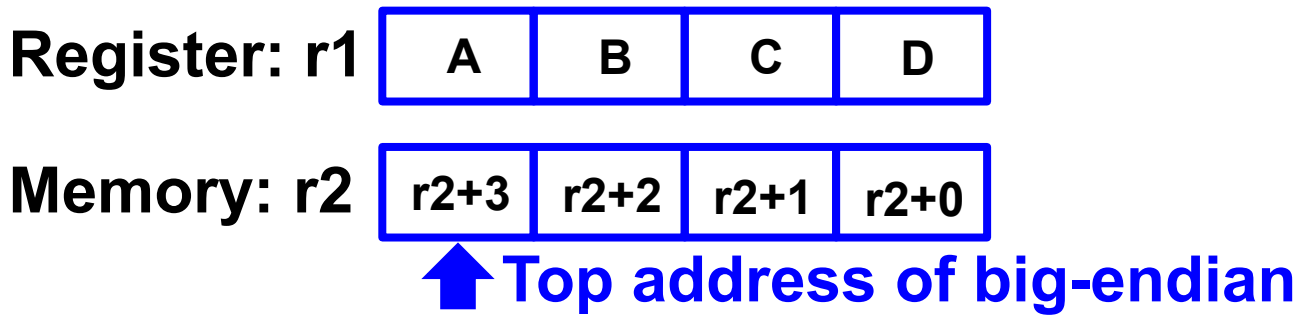


Difference in endian

Formative assessment

- Compile “for (i=0, j=0; i<10; i++) j+=2;” by hand.

- Describe little-endian instructions for storing four bytes in big-endian order



Formative assessment

●**Compile “for (i=0, j=0; i<10; i++) j+=2;” by hand.**

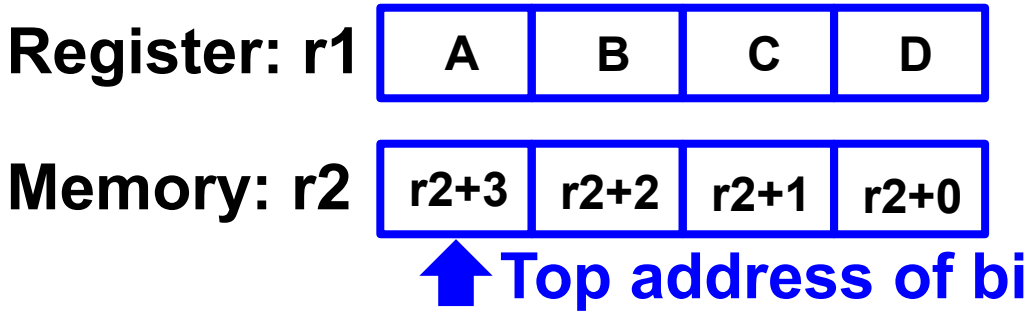
```
        add         0,0,r1    /* set 0 to i */
        add         0,0,r2    /* set 0 to j */
        branch      Label2
Label1: add         r2,2,r2   /* add 2 to j */
        add         r1,1,r1   /* add 1 to i */
Label2: subtract    r1,10
        branch on less than Label1
```

【Other Ans.】

```
        add 0,20,r2         /* always 20 */
```

Formative assessment

● Describe little-endian instructions for storing four bytes in big-endian order




```
store.byte    r1,(r2+3)      /* store lowest byte of r1 to [r2+3] */
srl           r1,8,r1        /* shift logical r1 by 8bits */
store.byte    r1,(r2+2)      /* store lowest byte of r1 to [r2+2] */
srl           r1,8,r1        /* shift logical r1 by 8bits */
store.byte    r1,(r2+1)      /* store lowest byte of r1 to [r2+1] */
srl           r1,8,r1        /* shift logical r1 by 8bits */
store.byte    r1,(r2+0)      /* store lowest byte of r1 to [r2+0] */
```

Goal of today

Q1. Semiconductors are imperfect switches. What is the mechanism for making complete switches?
半導体は不完全なスイッチである.完全なスイッチにできる仕組みは?

Q2. What are three main components of a computer.
コンピュータの主要構成要素を3つ挙げよ

Q3. What is the background of CISC in the past, and RISC now?
昔はCISC,今はRISCを指向する背景は?

Q4. In network, which order is correct? Big endian or little endian? Which order in Intel and ARM?
ビッグエンディアンとリトルエンディアン,ネットワークに流して良いのはどっち? Intel,ARMはどっち?

Q5. getname () {char buf [80]; gets (buf); ...} How is this program attacked?
getname() {char buf[80]; gets(buf);...} このプログラムが乗っ取られる仕組みは?

Q6. Your program is correct. But 100 times slower than colleague's program. What should you do?
あなたのプログラムは無駄がなく結果も正しい.でも同僚のプログラムより100倍遅い.気付くべき点は?

CA0303:Memory-space, stack and buffer overflow

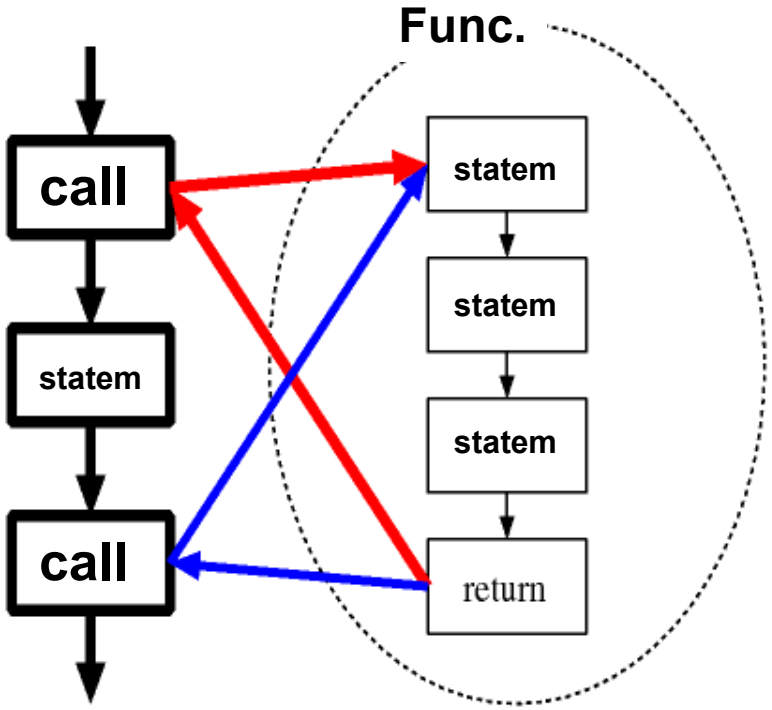
Function call and return

Function call:

```
sub (args);  /* function call */
:
return (value);  /* return with value */
```

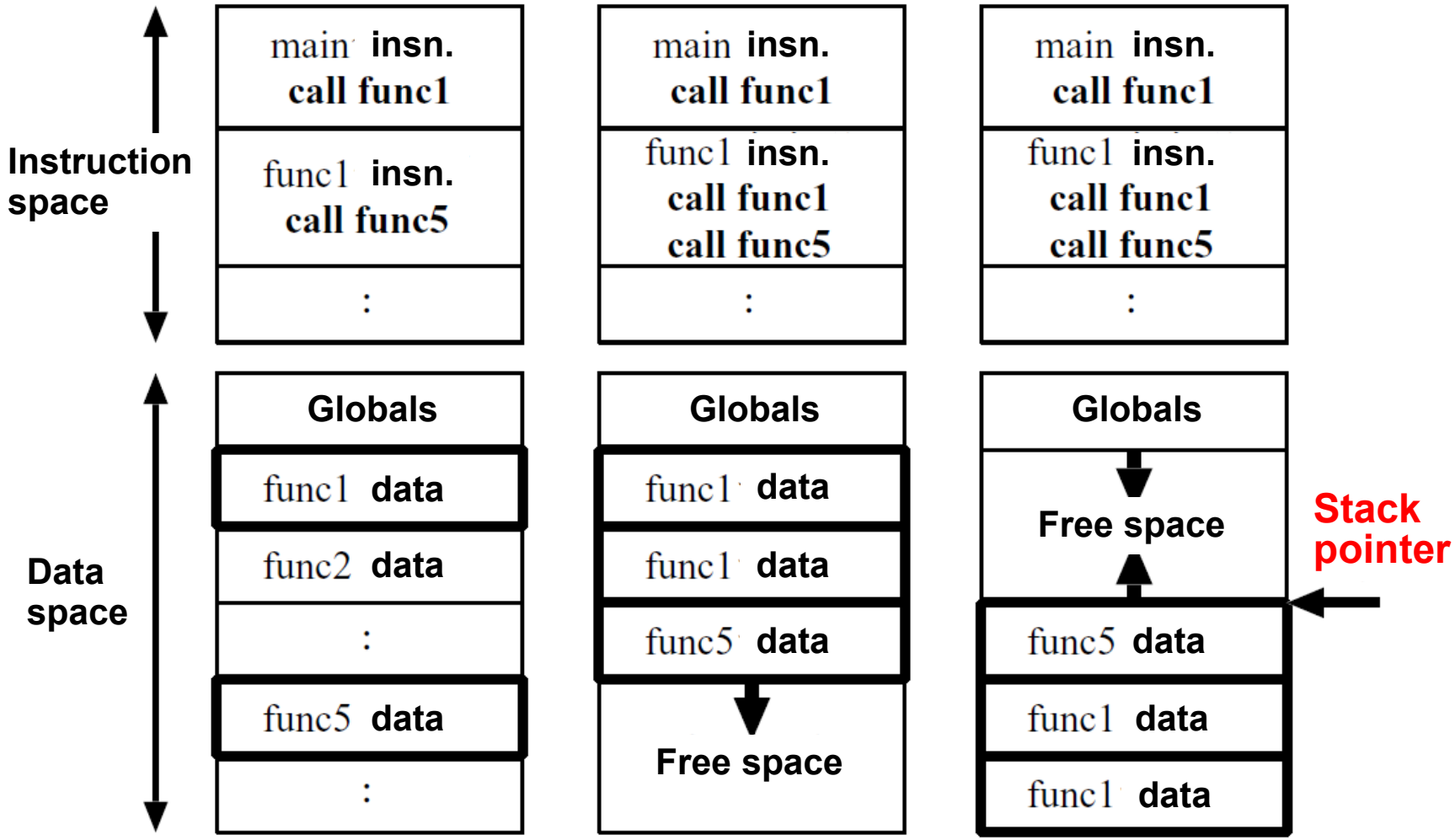
ARM instructions

```
bl   sub  …  save next address to lr (link-register)
:
sub:
:
bx  lr      … return to saved address
```



- Nested call destroys previous value in link-register.
- Systematic procedure to save/restore link-register is important.

How to locate local variables in each function

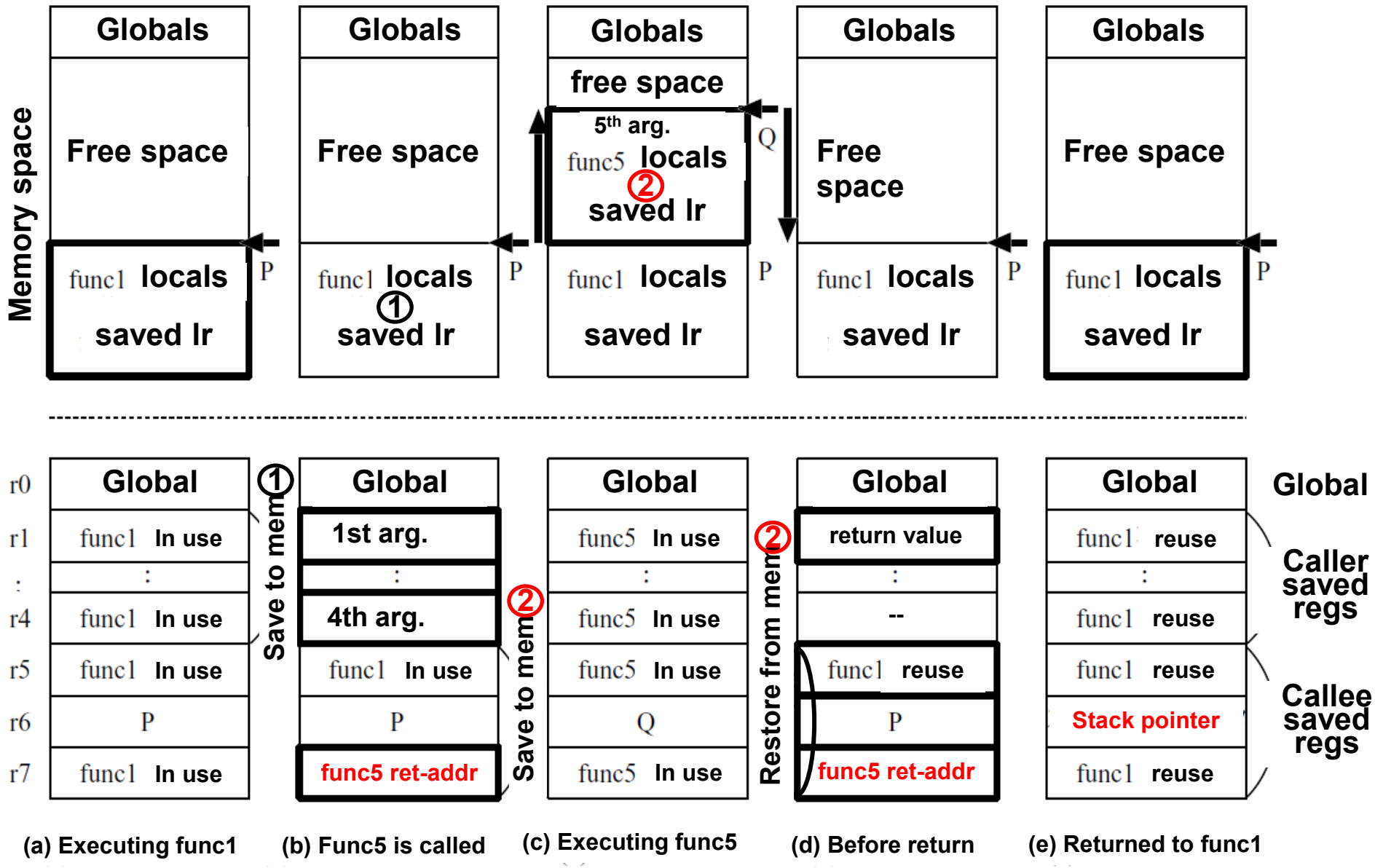


(a) Static allocation for each function

(b) In-order allocation for each call

(c) Relative allocation for each call

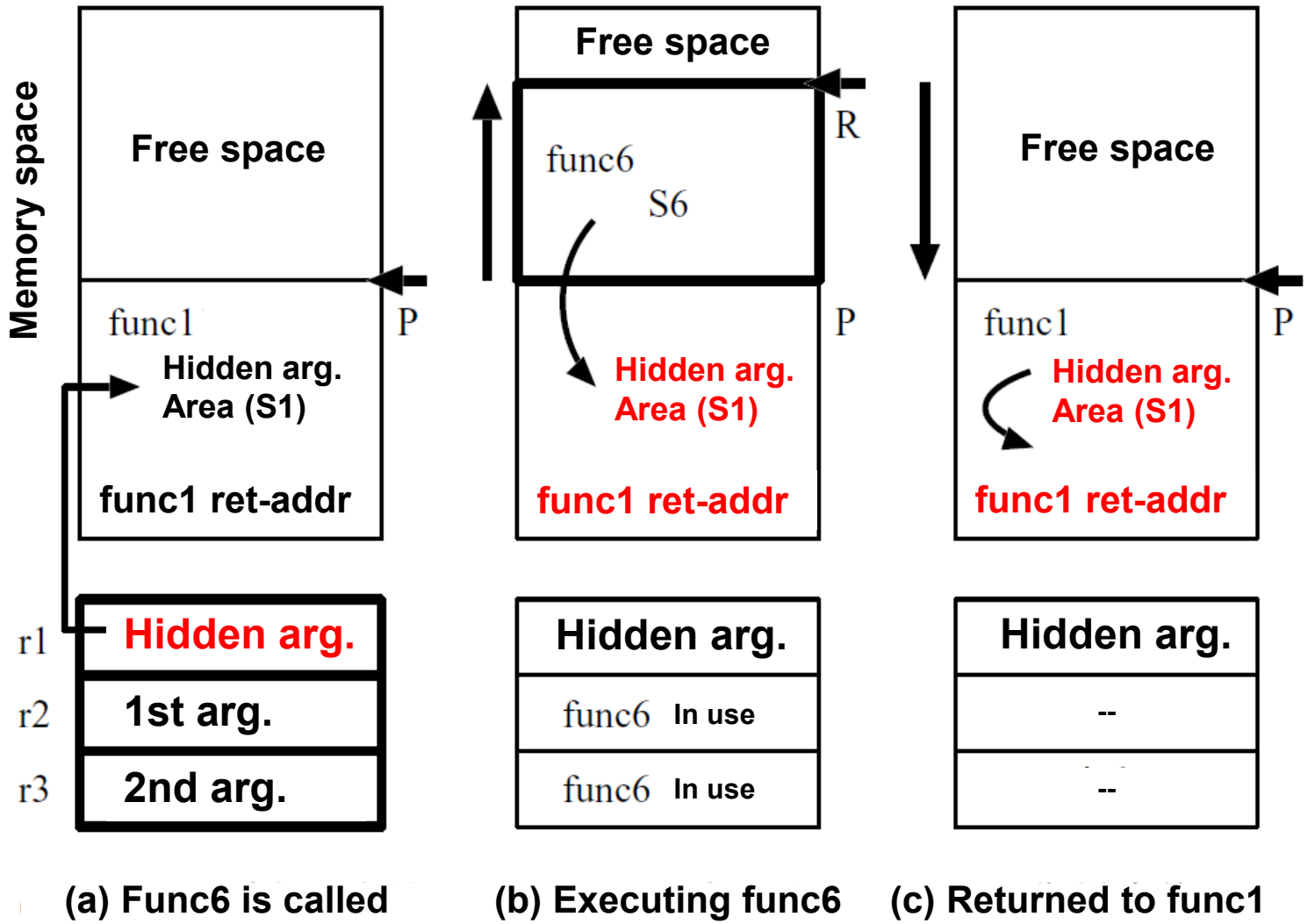
Structure of call stack



(a) Executing func1 (b) Func5 is called (c) Executing func5 (d) Before return (e) Returned to func1

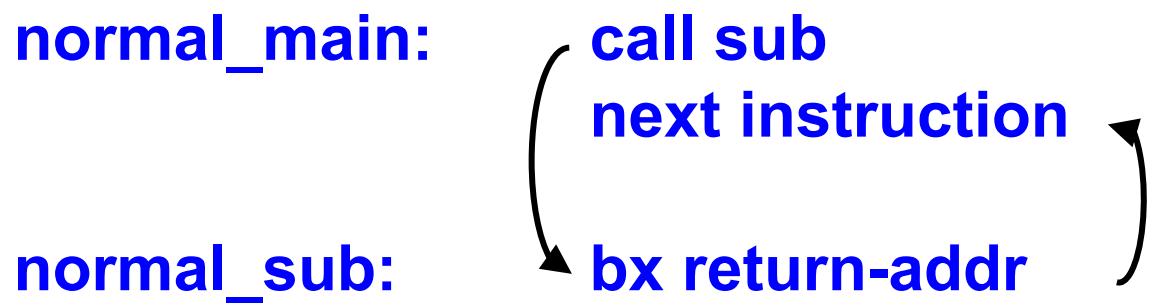
Hidden argument for multi-word return value

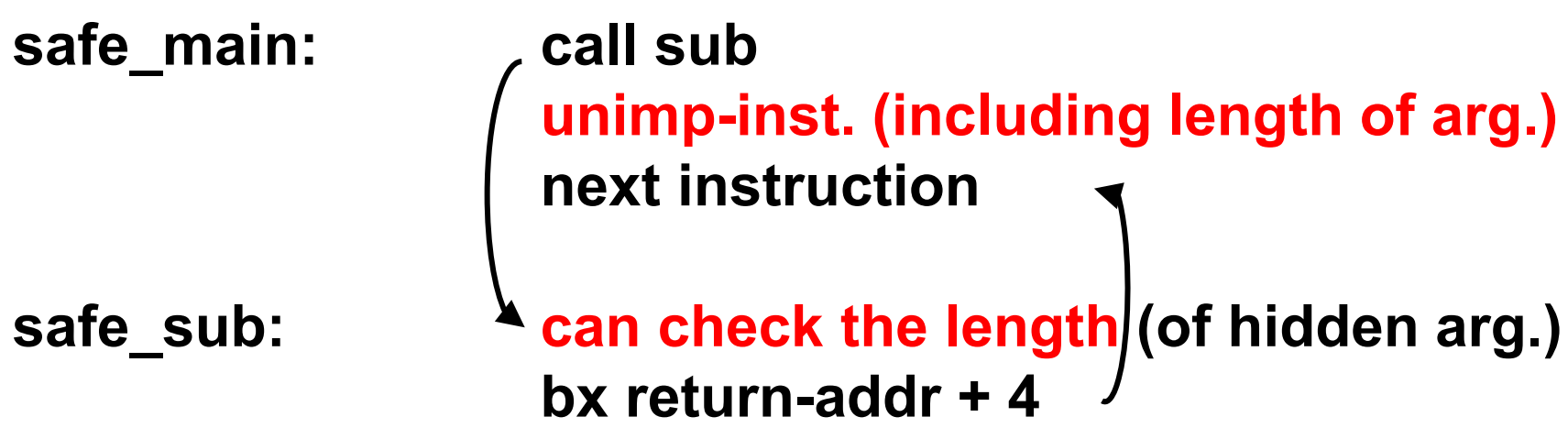
```
struct s {
  int m[100];
};

struct s func6() {
  struct s S6;
   :
  return(S6);
}

func1() {
  struct s S1;
  S1 = func6();
}
```



Safe way to pass hidden argument

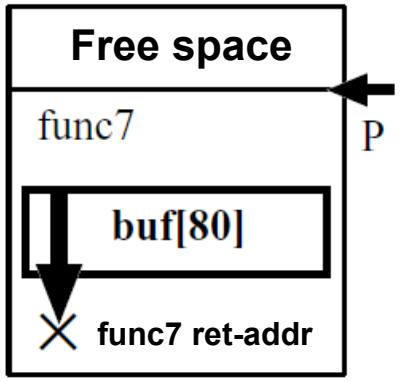
- Hidden arguments (pointer) is required to get return value with multi-words.
- Overrun of the hidden argument destroys saved return-address.





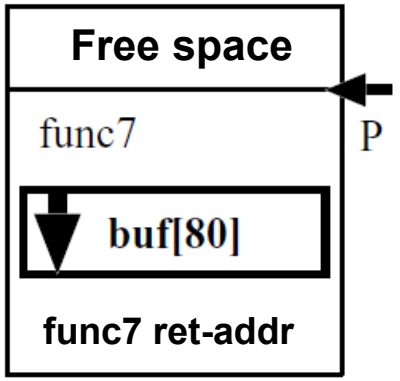
Buffer overflow attack and defense

```
func7() {
  char buf[80];
  gets(buf);
  :
}
```



(a) Without checking length

```
func7() {
  char buf[80];
  fgets(buf, 80, stdin);
  :
}
```



(b) With checking length

Buffer overflow attack and defense


```
main() {
  printf("=== Before ===¥n");
  sub();
  printf("=== After ===¥n");
}
sub() {
  char s[10];
  gets(s);
  printf("input = ¥"%s¥"¥n", s);
}
```

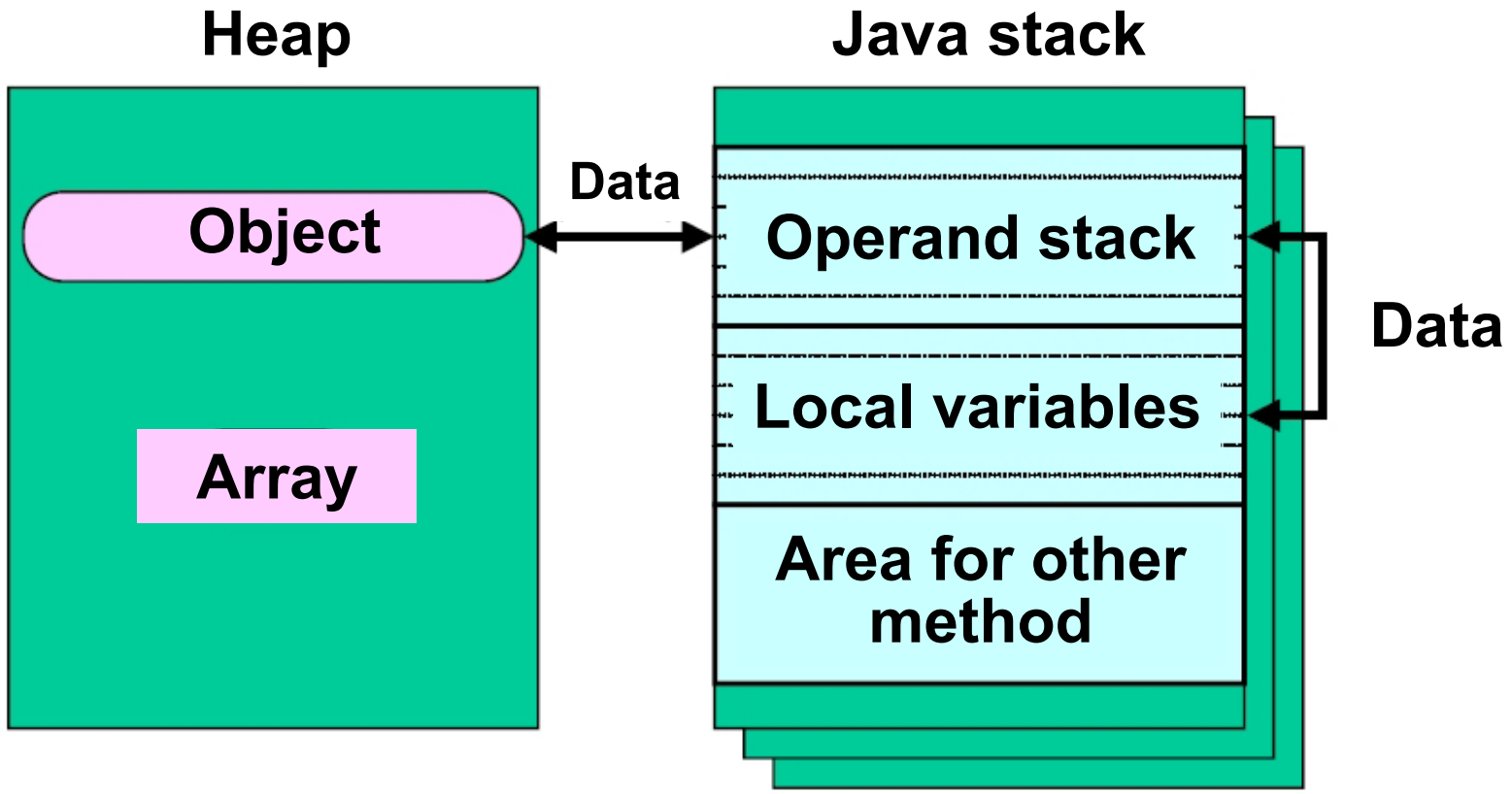
```
% cc a1.c -o a1
/tmp/ccxwZ.o(.text+0x4d): In function `sub':
: warning: this program uses gets(), which is unsafe.

% ./a1
=== Before ===
warning: this program uses gets(), which is unsafe.
naist
input = "naist"
=== After ===

% ./a1
=== Before ===
warning: this program uses gets(), which is unsafe.
naistnaistnaistnaistnaistnaistnaist
input = "naistnaistnaistnaistnaistnaistnaist"
Segmentation fault
```

Buffer overflow attack and defense


```
main() {
  printf("=== Before ===¥n");
  sub();
  printf("=== After ===¥n");
}
sub() {
  char s[10];
  fgets(s, 10, stdin);
  printf("input = ¥"%s¥"¥n", s);
}
```

```
% cc a2.c -o a2

% ./a2
=== Before ===
naist
input = "naist"
=== After ===

% ./a2
=== Before ===
naistnaistnaistnaistnaistnaistnaist
input = "naistnais"
=== After ===
```

Overview of Java VM (virtual machine)



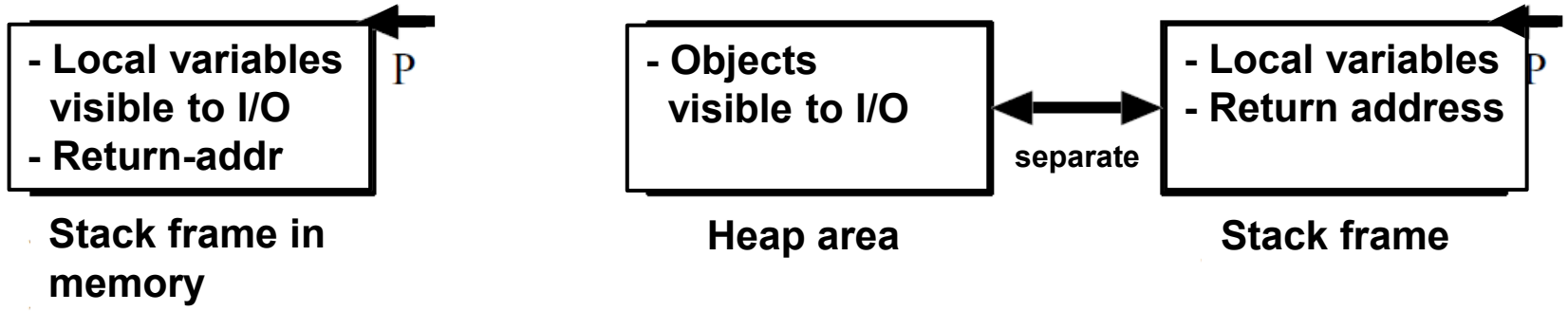
I/O is connected to heap.
Data should be moved to stack for calculation.
Garbage collection is required for releasing memory.

Fundamental problem in common processors

- Common processors have single memory space.

  Stack: both of local variables and return-address are allocated
  Object: I/O reachable data are allocated

- If these are mixed in a single memory space, return-address may be tampered.
- However, releasing memory-space is very fast.
  (just changing stack-pointer can do that)



(a) Normal memory usage (b) Java VM

Goal of today

Q1. Semiconductors are imperfect switches. What is the mechanism for making complete switches?
半導体は不完全なスイッチである.完全なスイッチにできる仕組みは?

Q2. What are three main components of a computer.
コンピュータの主要構成要素を3つ挙げよ

Q3. What is the background of CISC in the past, and RISC now?
昔はCISC,今はRISCを指向する背景は?

Q4. In network, which order is correct? Big endian or little endian? Which order in Intel and ARM?
ビッグエンディアンとリトルエンディアン,ネットワークに流して良いのはどっち? Intel,ARMはどっち?

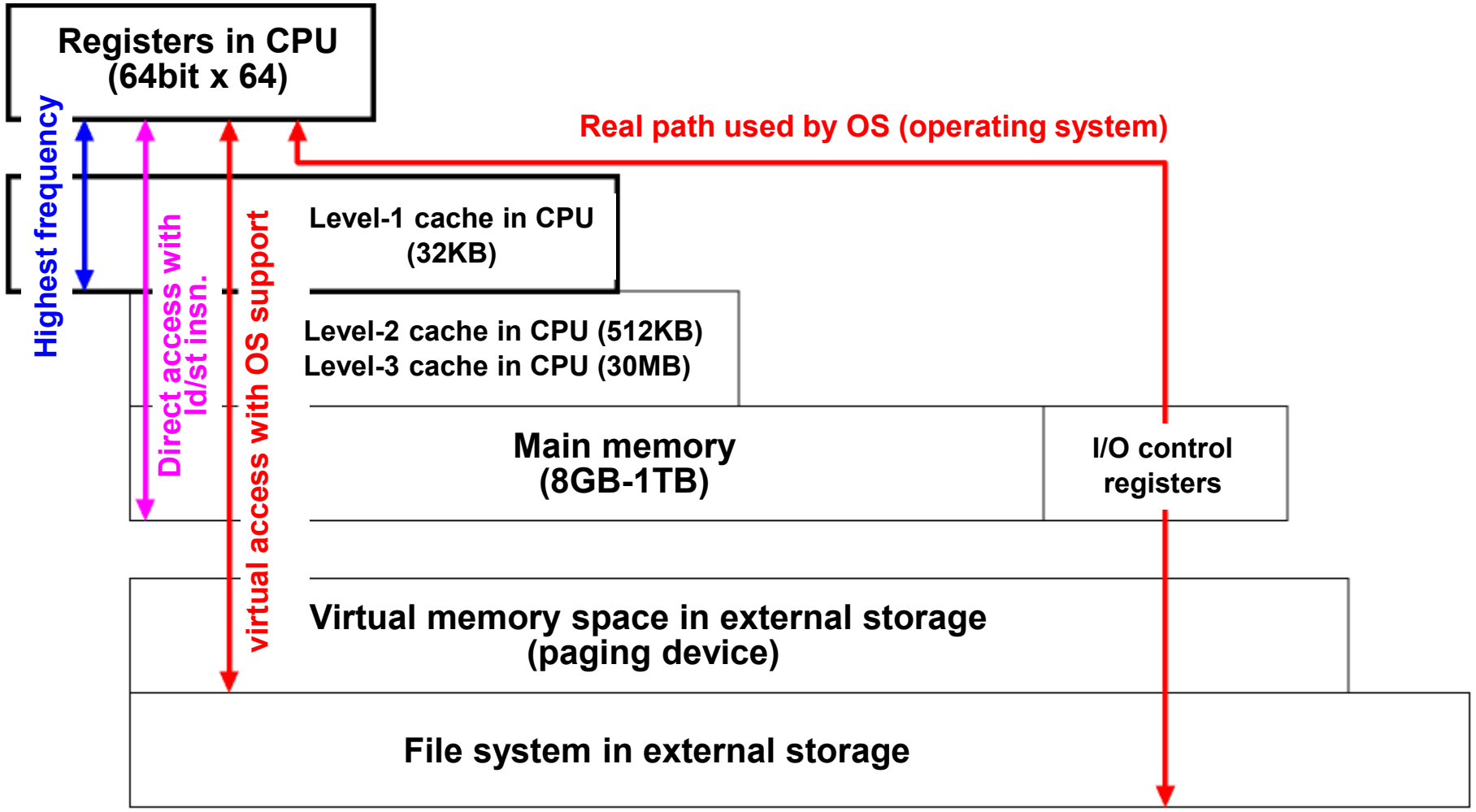
Q5. getname () {char buf [80]; gets (buf); ...} How is this program attacked?
getname() {char buf[80]; gets(buf);...} このプログラムが乗っ取られる仕組みは?

Q6. Your program is correct. But 100 times slower than colleague's program. What should you do?
あなたのプログラムは無駄がなく結果も正しい.でも同僚のプログラムより100倍遅い.気付くべき点は?

CA0702:Cache memory and execution speed of programs

Memory Hierarchy

Too slow without cache memory



Cache is not almighty

Spatial locality:

Memory location near past accessed tend to be accessed again.

Temporal locality:

Memory location near recently accessed tend to be accessed for a while.

That means following programs run very slow speed

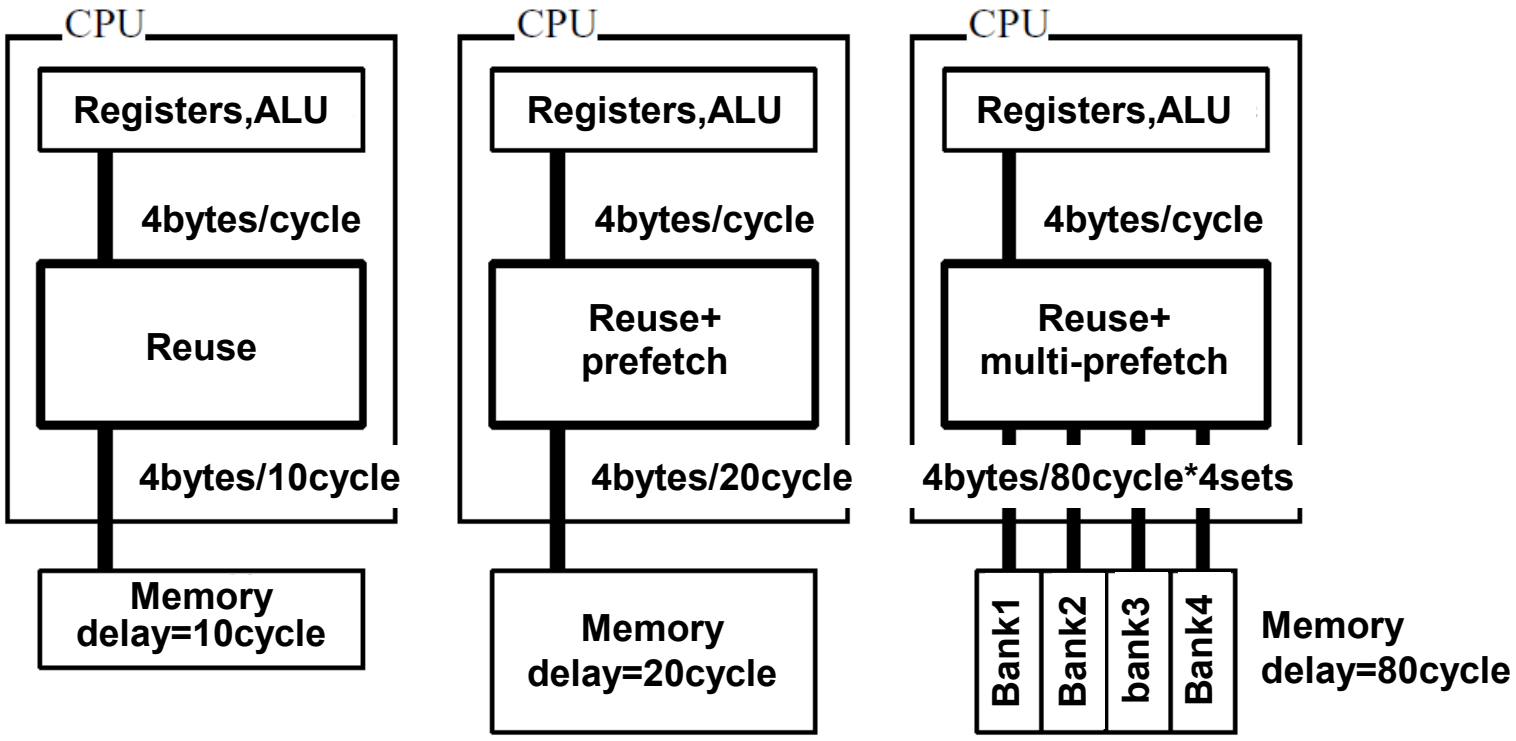
- Working set > capacity of cache
- Sequential access with no-reuse

Which cache can execute your program in fastest speed?

- Large but slow cache
- Small but fast cache
- Combination of multi-level cache

Ans. No one can recommend.
It depends on the skill of the programmer.

Evolution of cache to cover slow memory

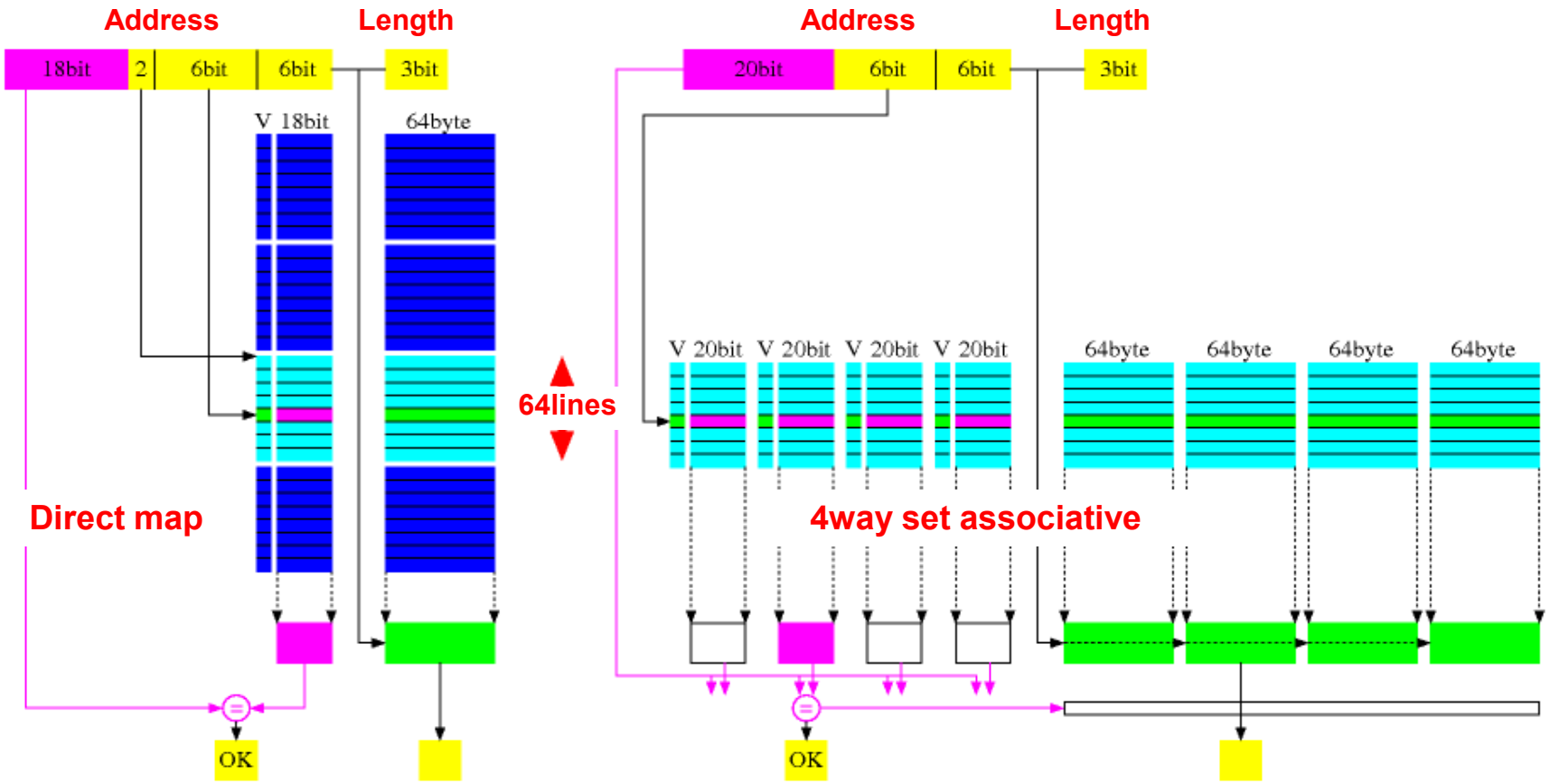


(a) Simple structure (b) Reuse+prefetch (c) Reuse+multiple-prefetch

Inside of cache

Memory address has three fields

- **High-bits for tag**
- **Mid-bits for line-number**
- **Low-bits for offset in each line**



Type of cache (write-through)

Write-through cache:

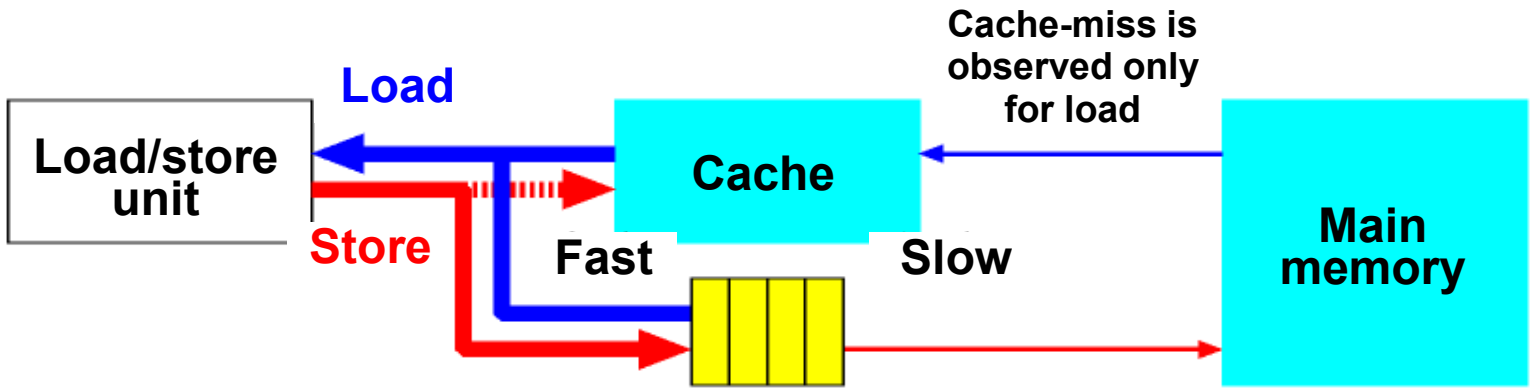
Store data to both of cache and memory simultaneously

× Store-buffer is required

○ Replacement is simple

Continuous store gives heavy pressure to store buffer.

- **If throughput from store buffer to memory is poor, the performance is significantly degraded.**



Type of cache (write-back)

Write-back cache:

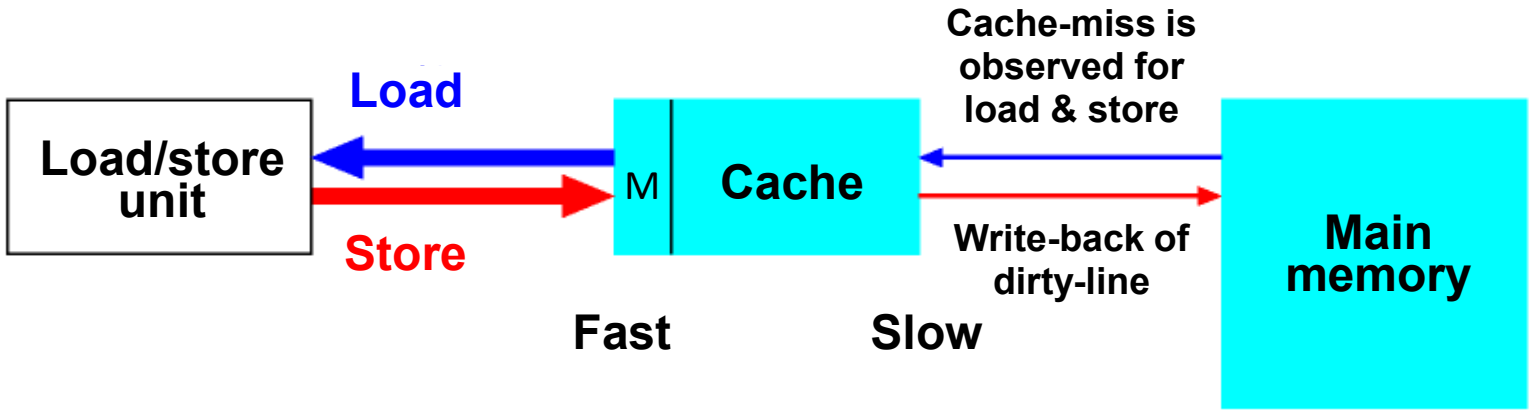
Store data only to cache, written back to memory later

○ high-speed for multiprocessor system

× Replacement is complicated

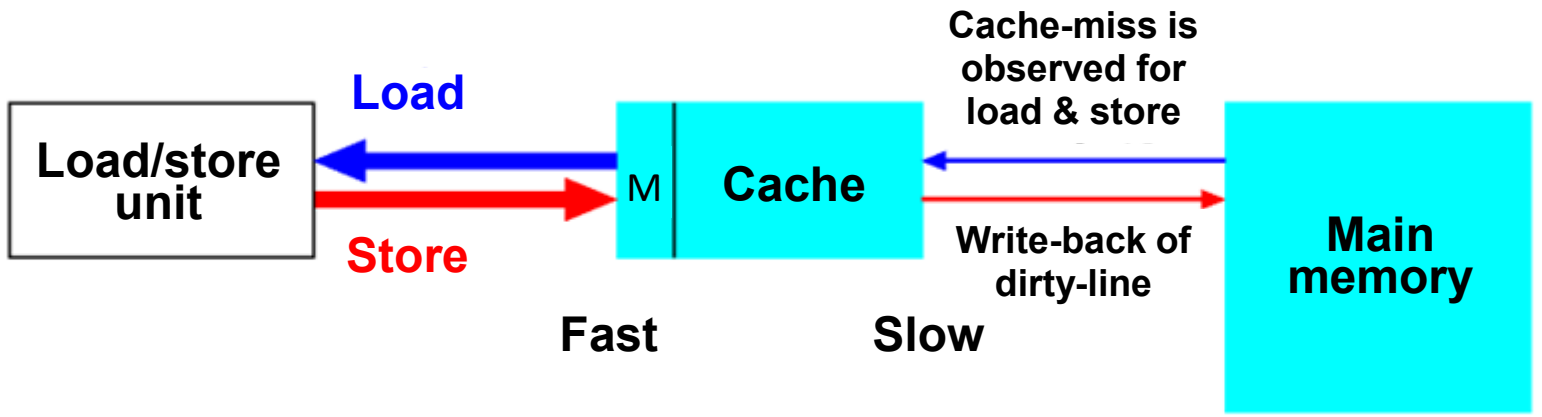
Can save power consumption of memory bus.

Can save traffic of memory bus.



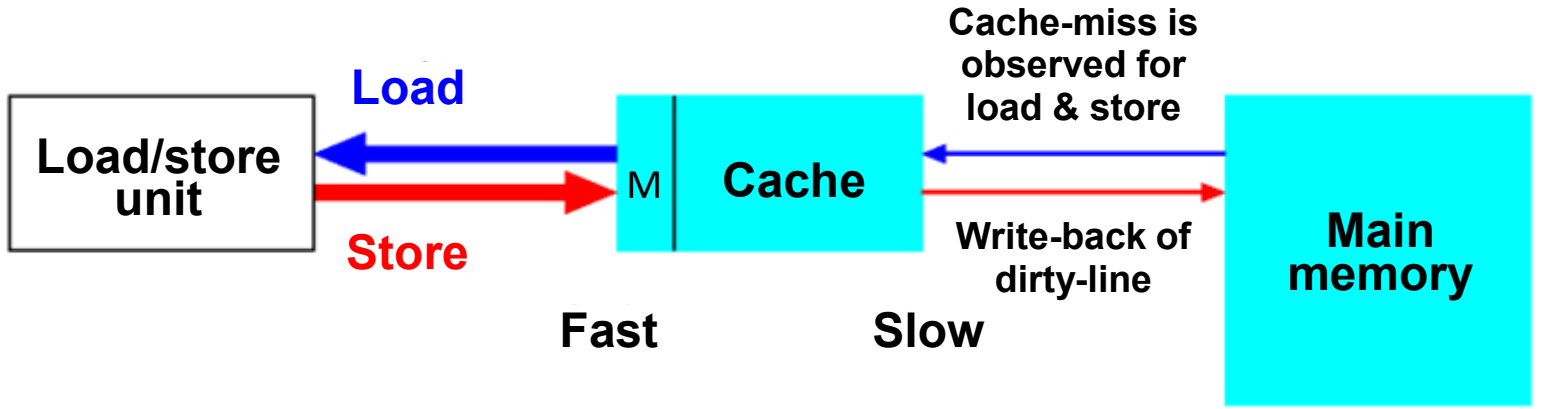
Formative assessment

● Describe the procedure when store misses cache.
(consider both of the case M=0 and M=1)



Formative assessment

● Describe the procedure when store misses cache. (consider both of the case M=0 and M=1)



Let's assume "store addr-A" misses cache, and corresponding cache-line has valid data of addr-B.

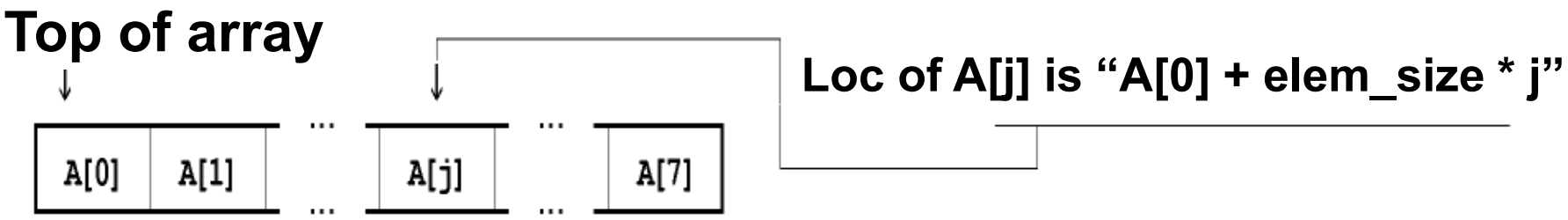
- If M=0, the cache-line has the same value as in addr-B.
  1. The line can be discarded immediately.
  2. Send read-request for addr-A to memory.
  3. Arrived data is stored in the line.
- If M=1, the cache-line has new value and memory has old value.
  1. Send write-request for addr-B to memory.
  2. Send read-request for addr-A to memory.
  3. Arrived data is stored in the line.

4. M is changed to 1.

Programming language and data cache

Alignment of data-structure and cache-structure (1D-array)

- Architectural important issue is layout of data-structure.



In case of C:

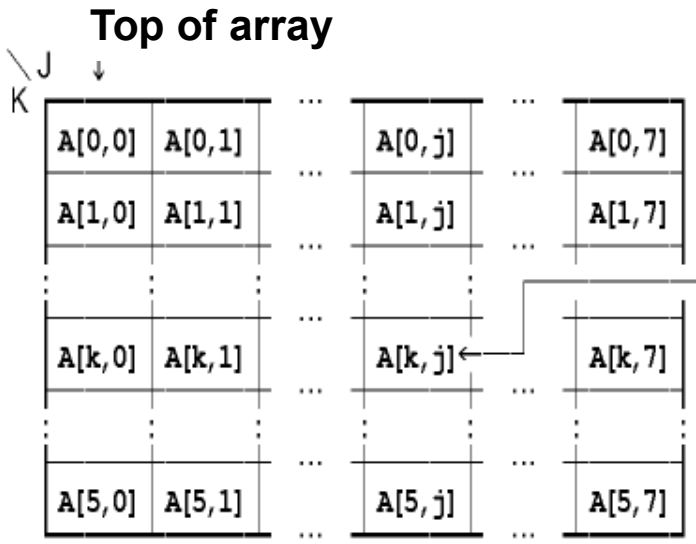
```
int i;
int A[1000];                  /* 0 .. 999
for (i=0; i<1000; i++) {      /* 0 .. 999
  A[i] = xxxx;
}
```

In case of FORTRAN:

```
INTEGER I
INTEGER A(1000)
DO 10 I=1,1000        ··· 1 .. 1000
  A(I) = xxxx         ··· loc of A[I] is “A[1] + elem_size * (I-1)”
10 CONTINUE
```

Alignment of data-structure and cache-structure (2D-array)

- **Location of data depends on programming language.**



Loc of A[k,j] is "A[0,0] + elem_size * (num_j * k + j)"
or "A[0,0] + elem_size * (num_k * j + k)"

In case of C:

```
for (i=0; i<1000; i++)   /* A[0][0] ... A[999][999]
  for (j=0; j<1000; j++)/* Inner-most loop should right-side index
    A[i][j] = xxxx;  /* A[0][0] A[0][1] … A[999][998] A[999][999]
```

In case of FORTRAN:

```
DO 10 I=1,1000
  DO 10 J=1,1000     ··· Inner-most loop should left-side index
    A(J,I) = xxxx    ··· A[1][1] A[2][1] … A[999][1000] A[1000][1000]
10 CONTINUE
```

Which one is faster?

```
double A[2048][2048];
double B[2048][2048];

main()
{
  int i, j, k;
  for (i=0; i<64; i++) {
    for (j=0; j<2048; j++) {
      for (k=0; k<2048; k++)
        A[k][j] = B[k][j]*2.0+1.0;
} } }
```

time: 10.94 sec

```
double A[2048][2048];
double B[2048][2048];

main()
{
  int i, j, k;
  for (i=0; i<64; i++) {
    for (j=0; j<2048; j++) {
      for (k=0; k<2048; k++)
        A[j][k] = B[j][k]*2.0+1.0;
} } }
```

time: 0.6 sec

Summary

- Q1. Semiconductors are imperfect switches. What is the mechanism for making complete switches?
  半導体は不完全なスイッチである.完全なスイッチにできる仕組みは?
- Q2. What are three main components of a computer.
  コンピュータの主要構成要素を3つ挙げよ
- Q3. What is the background of CISC in the past, and RISC now?
  昔はCISC,今はRISCを指向する背景は?
- Q4. In network, which order is correct? Big endian or little endian? Which order in Intel and ARM?
  ビッグエンディアンとリトルエンディアン,ネットワークに流して良いのはどっち? Intel,ARMはどっち?
- Q5. getname () {char buf [80]; gets (buf); ...} How is this program attacked?
  getname() {char buf[80]; gets(buf);...} このプログラムが乗っ取られる仕組みは?
- Q6. Your program is correct. But 100 times slower than colleague's program. What should you do?
  あなたのプログラムは無駄がなく結果も正しい.でも同僚のプログラムより100倍遅い.気付くべき点は?

Download the template and submit through UNIPA.

http://archlab.naist.jp/Lectures/ARCH/ca01_0301_0303_0702/ca010307e.docx

in http://archlab.naist.jp/Lectures

That’s all for today