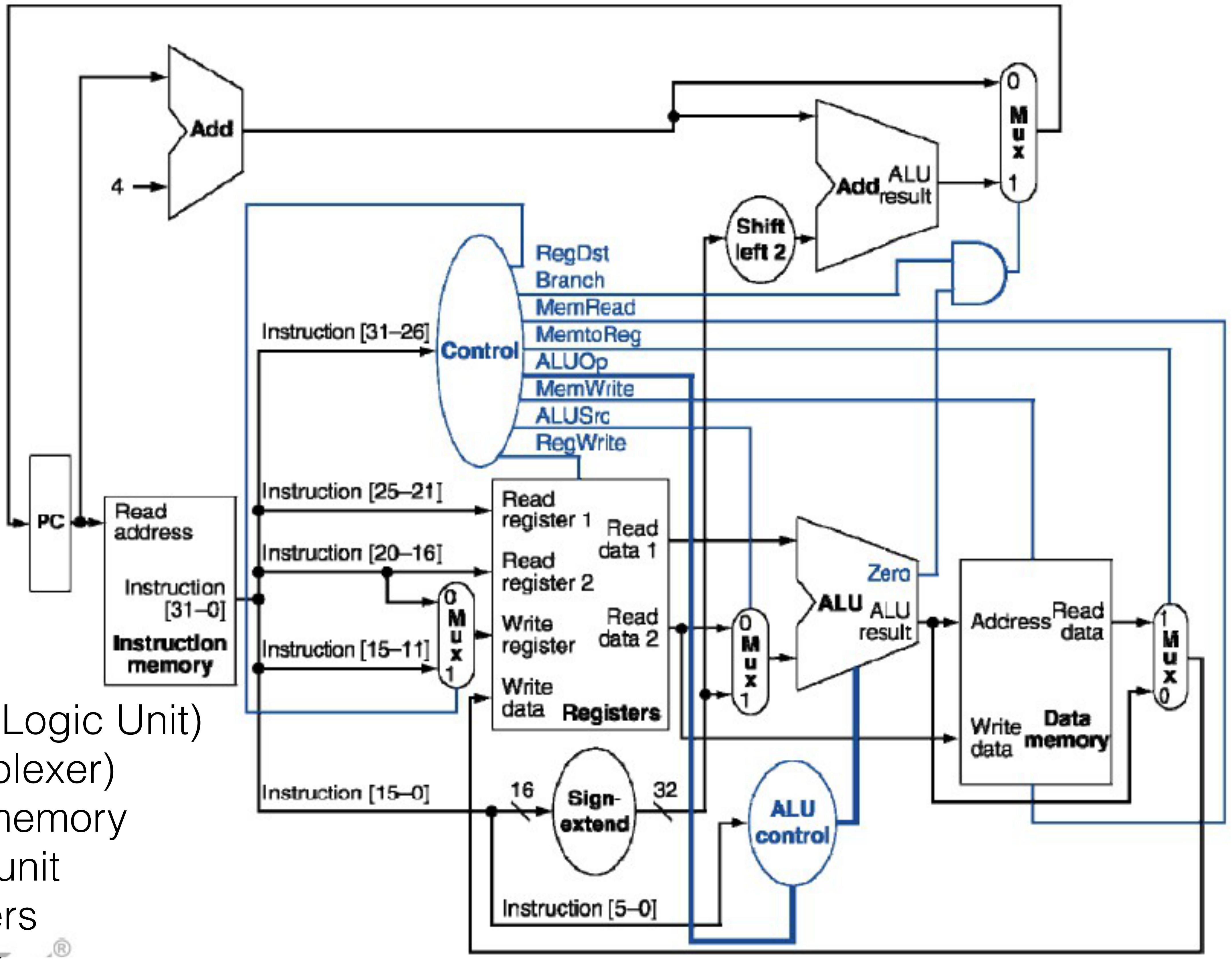
EE116C/CS151B Fall 2017

Instructor: Professor Lei He
TA: Yuan Liang



ALU (Arithmetic Logic Unit)
MUX (multiplexer)
Instruction memory
Controll unit
Registers
Clock
….

Review

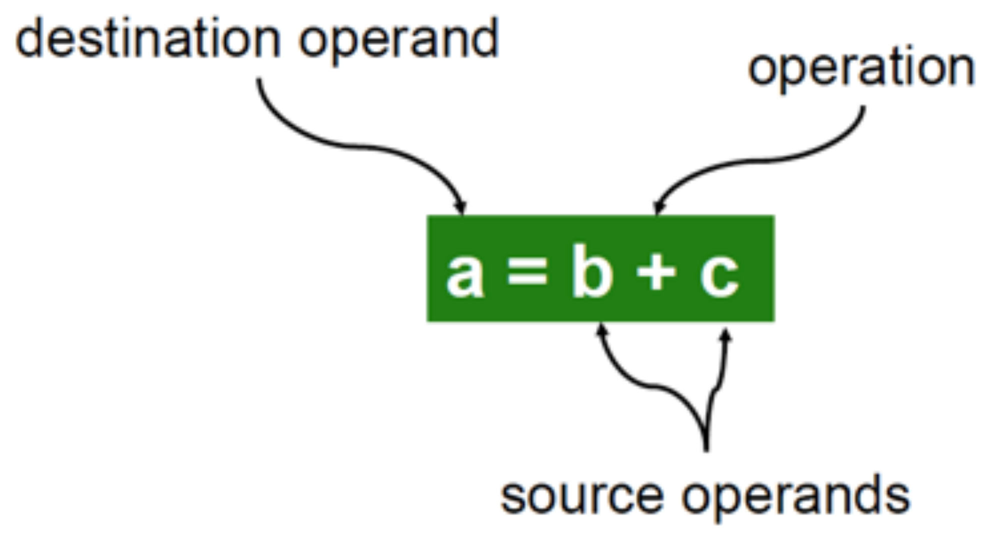
Definition of Instruction set architecture (ISA, or IS, or Instruction set) from Wiki:

An instruction set architecture (ISA) is a group of commands for a CPU in machine language.

It is an abstract model of a computer.

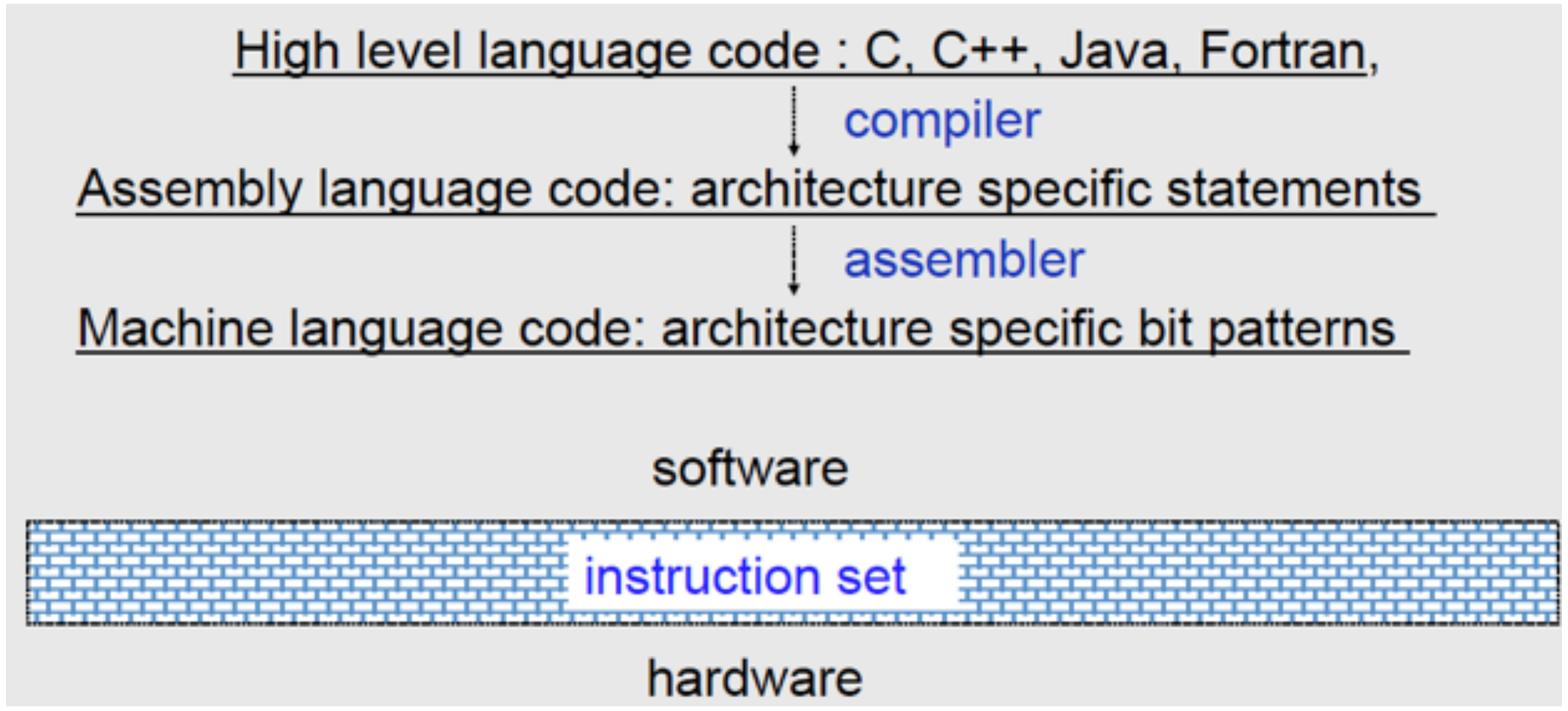
Different computers have different instruction sets.

Format:



Review

Serves as an interface between software and hardware


High level language code : C, C++, Java, Fortran,
compiler
Assembly language code: architecture specific statements
assembler
Machine language code: architecture specific bit patterns
software
instruction set
hardware

Review

Types of ISA:

CISC (Complex Instruction Set Computers)
large # of instructions
Advantage:
   many specialized instructions
   optimized at the instructions level
Disadvantage:
   various CPI
Intel’s x86 (1978)

RISC (Reduced Instruction Set Computers)
relatively fewer instructions
Advantage:
   same CPI -> enable pipelining and parallelism
Disadvantage:
   more instructions.
   more registers for intermediate results.
MIPS

Review

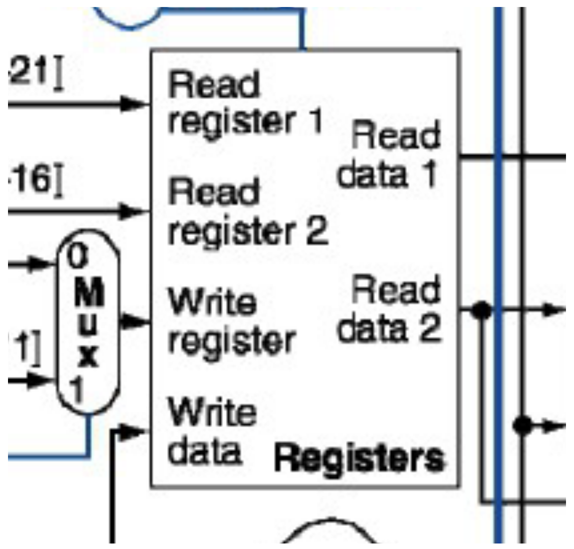
Registers in MIPS:

What is register:
Inside processor, use for frequently accessed data and fast calculation.

How many registers in MIPS:
32

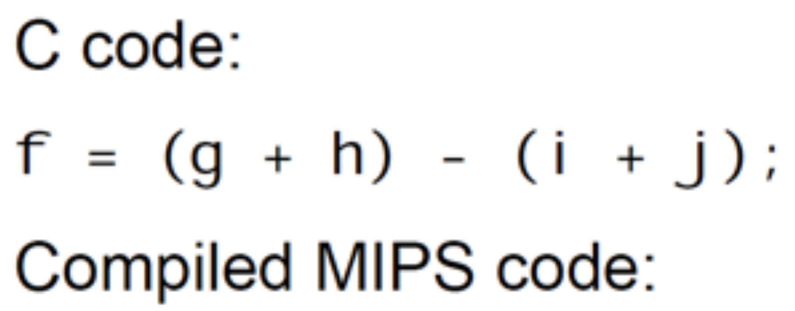
How large are them:
32bit

How they work:
Load values from memory into registers
Store result from register to memory



Review

Registers in MIPS:



C code:

```
f = (g + h) - (i + j);
```

Compiled MIPS code:

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

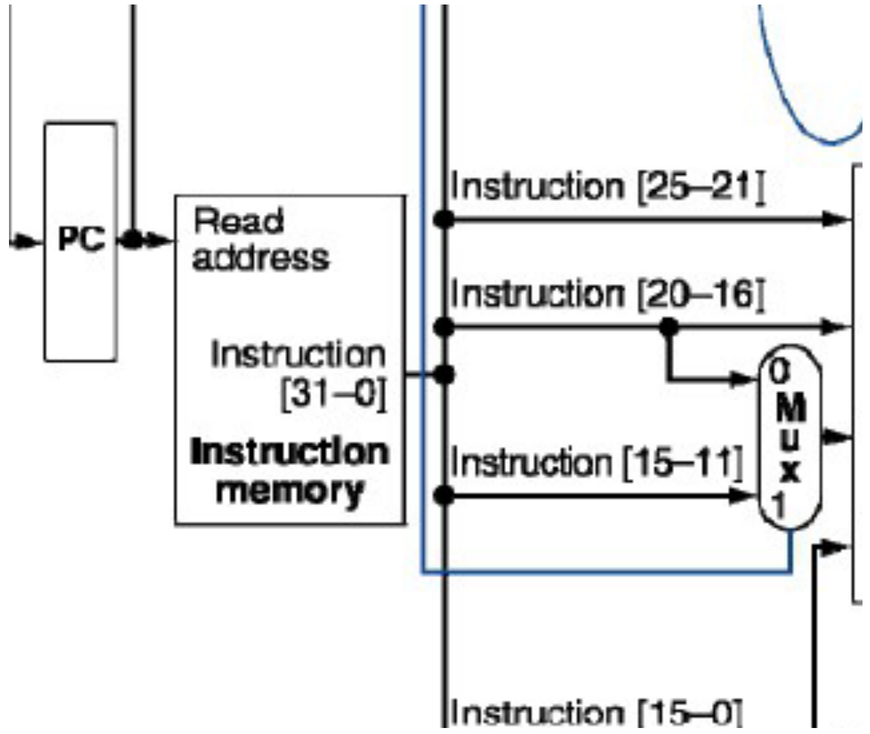
Review

Instructions in MIPS

Instructions formats are hard encoded in binary

For MIPS instructions

Encoded as 32-bit instruction words
Small number of formats encoding operation code (opcode), register numbers, …



Review

Instructions in MIPS

R-format Instructions

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

op: operation code (opcode)

rs: first source register number

rt: second source register number

rd: destination register number

shamt: shift amount (00000 for now)

funct: function code (extends opcode)

```
add $t0, $s1, $s2
```

0	17	18	8	0	32

$00000010001100100100000000100000_2$

Review

Instructions in MIPS

I-format Instructions

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

op: operation code (opcode)
rs: first source register number
rt: destination or source register number
Constant: -2^{15} to $+2^{15}-1$
Address: offset added to base address in rs

lw $s1, 100($s2)

↙

35	**18**	**17**	**100**

beq $s1, $s2, 100

↙

4	**17**	**18**	**25**

Review

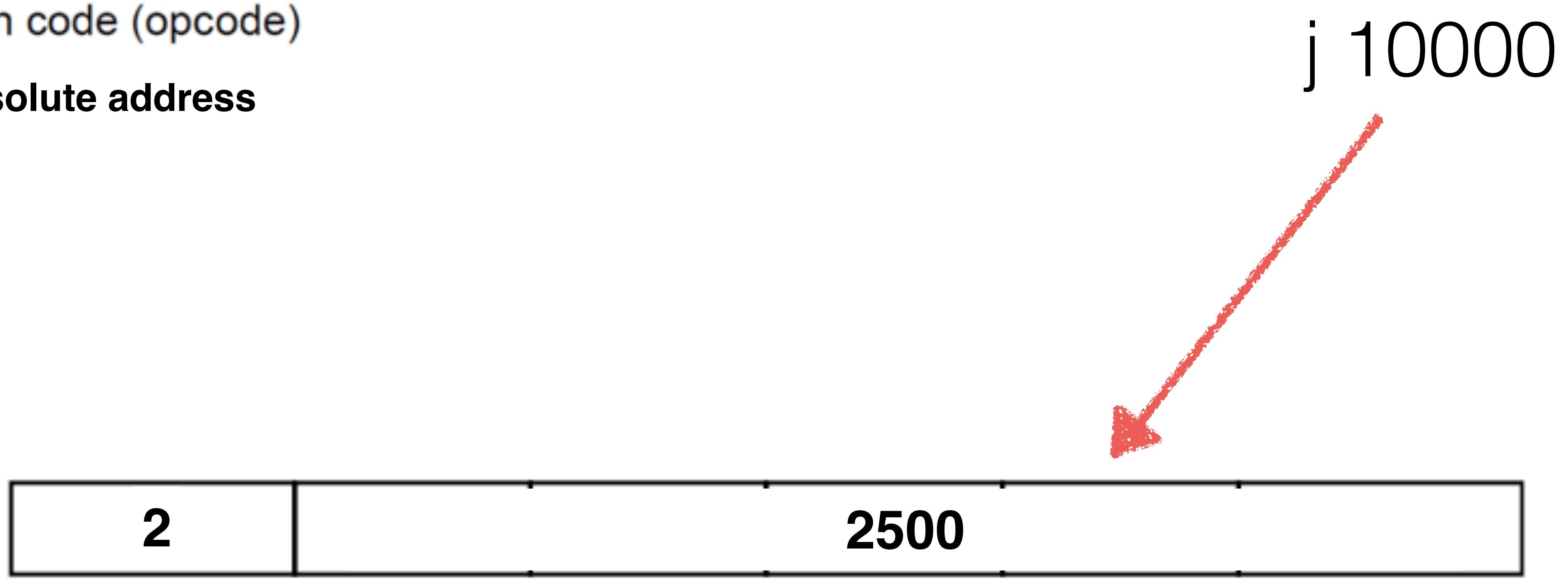
Instructions in MIPS

J-format Instructions

op	address
6 bits	26 bits

op: operation code (opcode)

Address: absolute address



Review

Instructions in MIPS

How does processor know what kind of instructions it is?

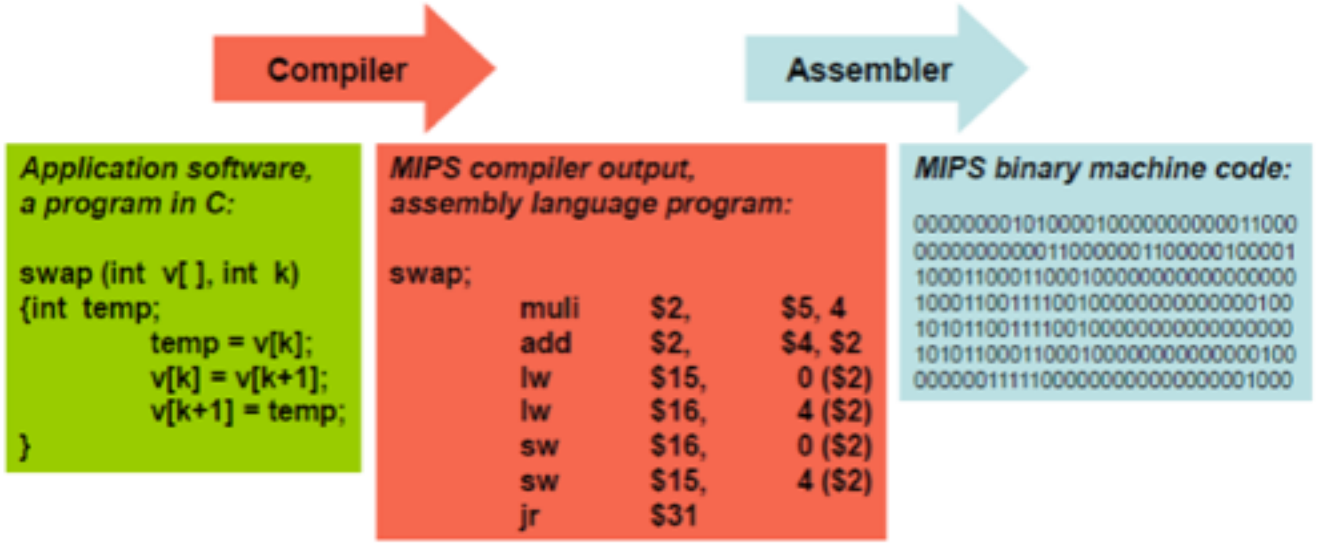
And how to parse it.

Review

- arithmetic
  - add, subtract, multiply, divide, ...
- logical
  - and, or, shift left, shift right, ...
- data transfer
  - load word, store word
- conditional branch
  - beq & bne are PC-relative, since most targets are nearby
- unconditional jump
  - jump, jump register, branch and link

Review

The big picture:



Application software, a program in C:

```
swap (int  v[ ], int  k)
{int  temp;
            temp = v[k];
            v[k] = v[k+1];
            v[k+1] = temp;
}
```

MIPS compiler output, assembly language program:

```
swap;
            muli      $2,       $5, 4
            add       $2,       $4, $2
            lw        $15,       0 ($2)
            lw        $16,       4 ($2)
            sw        $16,       0 ($2)
            sw        $15,       4 ($2)
            jr        $31
```

MIPS binary machine code:

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

What is ISA
ISA style
ISA for MIPS

ISA in processor

Sample Question 1

C code:

A[12] = h + A[8];

h in $s2, base address of A in $s3.

MIPS code?

Sample Question 1

1. Memory is and only is byte addressable.
first byte in memory: address = 0
second byte in memory: address = 1
etc.

2. Each register size in processor is 32-bit (4 byte, 1 word).
We always use a word as one unit.
Thus, A[0] = 4th byte in memory _ 3rd byte in memory _ 2nd byte in memory _ 1st byte in memory

3. when we fetch a word, we give the starting byte address, and the fetcher will give us the cascaded 4 byte content.
A[0] -> 1st byte content's address

Sample Question 1

A[0] -> 1st byte content's address -> 0
A[1] -> 5th byte content's address -> 4
…

A[8] -> 32nd byte content's address -> 32

..

A[i] -> i * 4 byte content's address -> 4i

Sample Question 1

```
lw  $t0, 32($s3)    # load word
add $t0, $s2, $t0
sw  $t0, 48($s3)    # store word
```

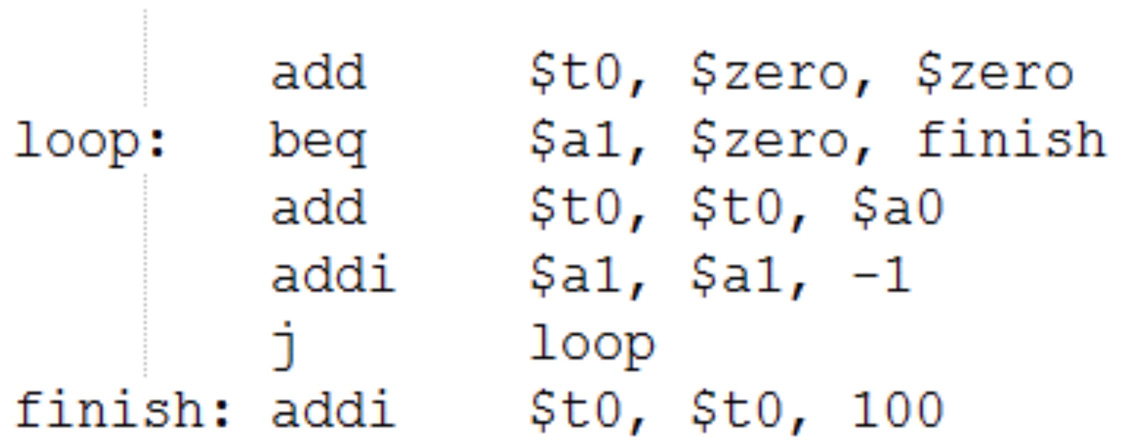
Sample Question 2

C code:

```
int sum = 0;
while (b != 0) {
  sum += a;
  b—;
}
sum = sum + 100;
```

MIPS code?

Sample Question 2



```
        add     $t0, $zero, $zero
loop:   beq     $a1, $zero, finish
        add     $t0, $t0, $a0
        addi    $a1, $a1, -1
        j       loop
finish: addi    $t0, $t0, 100
```

Sample Question 3

Initially, $s3, $s4, $s5 contains i, j, k. Let $s6 stores the base of A[].
Each element of A is a 32-bit word.

MIPS Instructions:

```
loop: add $t1, $s3, $s3
      add $t1, $t1, $t1
      add $t1, $t1, $s6
      lw $t0, 0($t1)
      add $s3, $s3, $s4
      bne $t0, $s5, exit
      j Loop
exit:
```

Sample Question 3

```
loop: add $t1, $s3, $s3
      add $t1, $t1, $t1
      add $t1, $t1, $s6
      lw $t0, 0($t1)
      add $s3, $s3, $s4
      bne $t0, $s5, exit
      j Loop
exit:
```

```
loop: t1 = 2i
      t1 = 4i
      t1 = s6 + t1
      t0 = A[i]
      i = i + j
      if (A[i] != k) goto exit
      goto loop
exit:
```

```
while (A[i] == k) i = i + j;
```