CS 4200/5200
Computer Architecture I

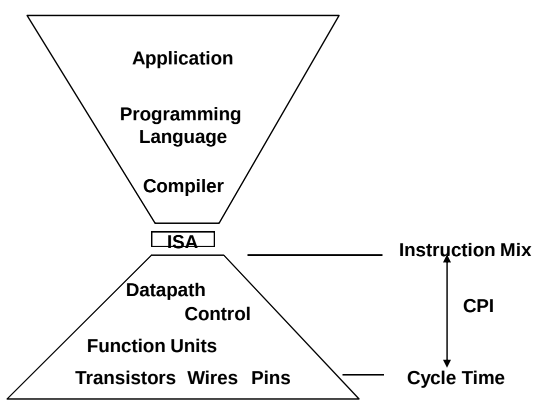
MIPS Instruction Set Architecture

Dr. Xiaobo Zhou

Department of Computer Science

Review: Organizational Trade-offs

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

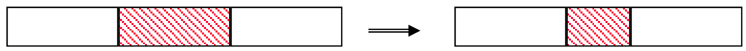


Review: Amdahl's Law

ExTime after improvement = ExTime unaffected + Extime affected / amount of improvement

Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$

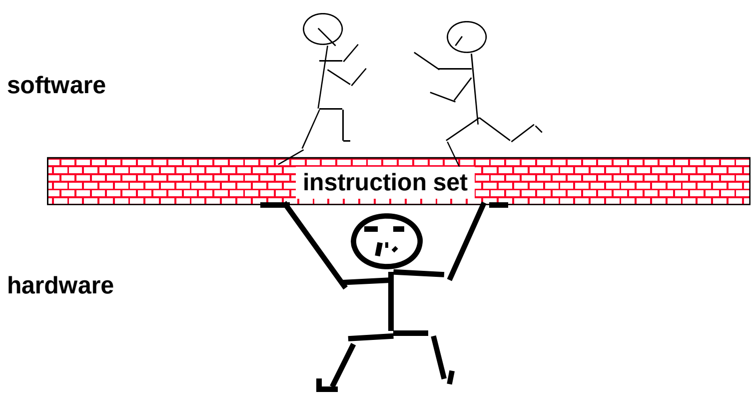


Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected then,

ExTime(with E) = ((1-F) + F/S) X ExTime(without E)

$$\text{Speedup(with E)} = \frac{\text{ExTime(without E)}}{((1\text{-}F) + F/S) \times \text{ExTime(without E)}} = \frac{1}{(1\text{-}F) + F/S}$$

Instruction Set Design



An instruction is a binary code, which specifies a basic operation (e.g. add, subtract, and, or) for the computer

- Operation Code: defines the operation type
- Operands: operation source and destination

Levels of Representation

High Level Language Program

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

Compiler

Assembly Language Program

```
lw  $15,  0($2)
lw  $16,  4($2)
sw  $16,  0($2)
sw  $15,  4($2)
```

Assembler

Machine Language Program

```
1000 1100 0100 1111 0000 0000 0000 0000
1000 1111 0101 1000 0000 1001 1100 0110
1010 1110 1010 1111 0101 1000 0000 1001
1010 1100 0000 1001 1100 0110 1010 1111
```

Machine Interpretation

?

Control Signal Specification

°
°

MIPS R2000 / R3000 Registers

- MIPS (NEC, SGI, Sony, Nintendo), a typical of instructions after 1980s.

- 32-bit machine --> Programmable storage
- 31 x 32-bit GPRs (R0 = 0)
- 32 x 32-bit FP regs
- HI, LO, PC
- Big Endian

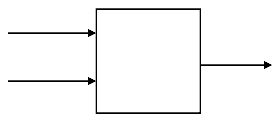
2^32 x 4 bytes

r0 0
r1
°
°
°
r31
PC
lo
hi

- Addressing modes:
  - register
  - immediate
  - displacement
  - PC-relative
  - pseudo-direct
- All instructions are 32-bit wide and must be aligned -- words must start at address that are multiple of 4.

MIPS arithmetic instructions

Instruction	*Example*	*Meaning*	*Comments*
add	add \$1,\$2,\$3	\$1 = \$2 + \$3	3 operands
subtract	sub \$1,\$2,\$3	\$1 = \$2 – \$3	3 operands
add immediate	addi \$1,\$2,100	\$1 = \$2 + 100	+ constant
sub immediate	subi \$1,\$2,100	\$1 = \$2 - 100	- constant



Example

E.g. f= (g+h) - (i+j),
assuming f, g, h, i, j be assigned to \$1, \$2, \$3, \$4, \$5

```
add $7, $2, $3    // register $7 contains g+h
add $8, $4, $5    // register $8 contains i+j
sub $1, $7, $8    // register $1 (f) gets the result
```

MIPS logic instructions

Instruction	*Example*	*Meaning*	*Comment*
and	**and $1,$2,$3**	**$1 = $2 & $3**	**Logical AND**
or	**or $1,$2,$3**	**$1 = $2 | $3**	**Logical OR**
xor	**xor $1,$2,$3**	**$1 = $2 ⊕ $3**	**Logical XOR**
nor	**nor $1,$2,$3**	**$1 = ~($2 |$3)**	**Logical NOR**
and imme.	**andi $1,$2, 100**	**$1 = $2 & 100**	**AND constant**
…			

x	y	x *and* y	x *or* y	x *xor* y	x *nor* y
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	1	0	0

MIPS data transfer instructions

Instruction	*Comment*
LW $1, 30($2)	**Load word**
LH $1, 40($3)	**Load half a word**
LB $1, 40($3)	**Load byte**
SW $3, 500($4)	**Store word**
SH $3, 502($2)	**Store half**
SB $2, 41($3)	**Store byte**

In MIPS64: LD (load double word)

LD $1, 30($2) load a double-word

SD $3, 500($4) store a double-word

Example

Assume A is an array of 100 words, and compiler has associated the variables g and h with the register $1 and $2. Assume the base address of the array is in $3. Translate

g = h + A[8]

```
lw $4, 8($3);            //  $4 <-- A[8]
add $1, $2, $4;
```

```
lw $4, 32($3);           //  $4 <-- A[8]
add $1, $2, $4
```

A[12] = h+A[8]   SW $1, 48($3)

Example

Assume A is an array of 100 words, and compiler has associated the variables g, h, and i with the register $1, $2, $5. Assume the base address of the array is in $3. Translate

g = h + A[i]

```
add  $6, $5, $5;       // $6 = 2i
add  $6, $6, $6;       //  $6 = 4i

add $4, $3, $6;        // $4 <---absolute mem. address of
                       // A[i], used as new base

lw   $7, 0 ($4);       // $7 = A[i]
add $1, $2, $7;        // g = h + A[i]
```

MIPS branch, jump, compare instructions

Instruction	***Example***	***Meaning***
branch on equal	beq $1,$2,100	if ($1 == $2) go to PC+4+100
	Equal test; PC relative branch	
branch on not eq.	bne $1,$2,100	if ($1!= $2) go to PC+4+100
	Not equal test; PC relative	
branch on eq. to 0	beqz $1,100	if ($1==0) go to PC+4+100
	Zero *test; PC relative (pseudo-instruction; using $0)*	

beq, bne, blt, blez, bgt, bgez

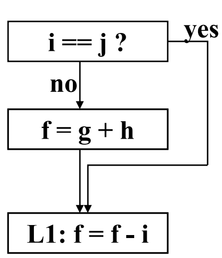
jump	j 10000	go to 10000
	Jump to target address	
jump register	jr $31	go to $31
	For switch, procedure return	
jump and link	jal 10000	$31 = PC + 4; go to 10000
	For procedure call	
set on less than	slt $1,$2,$3	if ($2 < $3) $1=1; else $1=0
	Compare less than; 2's comp.	
set less than imm.	slti $1,$2,100	if ($2 < 100) $1=1; else $1=0
	Compare < constant; 2's comp.	

Example

```
        if (i==j) go to L1;
        f = g+ h;
L1:     f = f - i;
```

Assuming f, g, h, i, j ~ $1, $2, $3, $4, $5

```
        beq $4, $5, L1
        add $1, $2, $3
L1:     sub $1, $1, $4
```

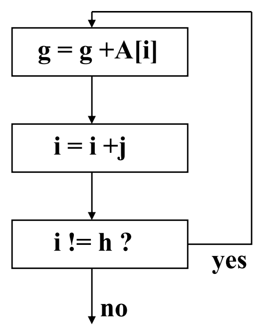


Example

Loop: g = g +A[i];
i = i+ j;
if (i != h) go to Loop:

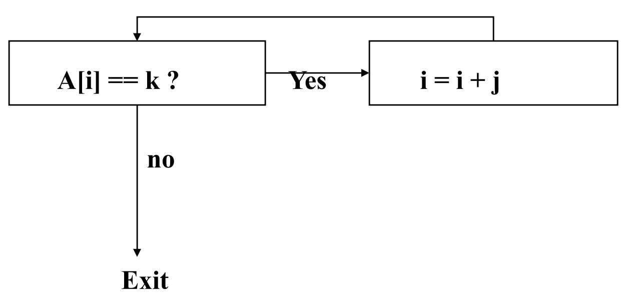
Assuming variables g, h, i, j ~ $1, $2, $3, $4 and base address of array is in $5

```
Loop:  add $7, $3, $3
       add $7, $7, $7
       add $7, $7, $5
       lw $6, 0($7)
       add $1, $1, $6    // g= g+A[i]
       add $3, $3, $4
       bne $3, $2, Loop;
```



Example

while (A[i]==k)
i = i+j;
Assume i, j, and k ~ $17, $18, $19 and base of A is in $3



Example

while (A[i]==k)
　　i = i+j;
Assume i, j, and k ~ $17, $18, $19 and base of A is in $3

```
Loop:  add $20, $17, $17
       add $20, $20, $20
       add $20, $20, $3

       lw $21,0($20)
       bne $21, $19, Exit     //exit loop if A[I] ! =k
       add $17, $17, $18
       j      Loop
Exit:
```


MIPS Fields

R-format:
Register (direct)

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

- **op: basic operation of the instruction, called opcode**
- **rs, the first register source operand**
- **rt: the second register source operand**
- **rd: the register destination operand**
- **shamt: shift amount**
- **funct: function, select the variant of the op field.**

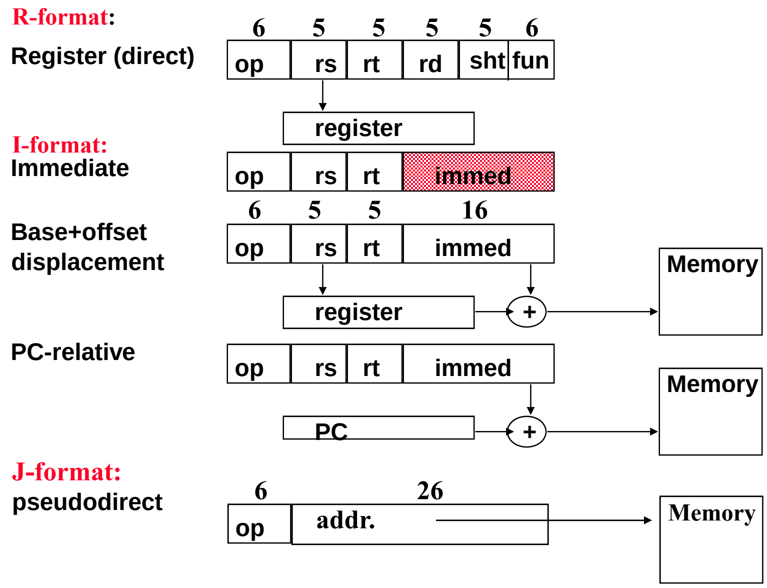
I-format:
Immediate

6	5	5	16
op	rs	rt	immediate

J-format:
jump

6	26
op	addr.

MIPS Addressing Modes/Instruction Formats



Example

while (A[i]==k)
  i = i+j;
Assume i, j, and k ~ $17, $18, $19 and base of A is in $3

Assume the loop is placed starting at loc 8000

```
Loop:  add $20, $17, $17
       add $20, $20, $20
       add $20, $20, $3
       lw $21,0($20)
       bne $21, $19, Exit
       add $17, $17, $18
       j       Loop
Exit:
```

8000:	0	17	17	20	0	32
	0	20	20	20	0	32
	0	20	3	20	0	32
	35	20	21	0		
	5	21	19	8 -> 2		
	0	17	18	17	0	32
	2	8000 -> 2000				

MIPS arithmetic & logical instructions

Instruction	*Example*	*Meaning*	*Comments*
add	add $1,$2,$3	$1 = $2 + $3	3 operands;
subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands;
add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant;
add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands;
subtract unsigned	subu $1,$2,$3	$1 = $2 – $3	3 operands;
add imm. unsign.	addiu $1,$2,100	$1 = $2 + 100	+ constant;
and	and $1,$2,$3	$1 = $2 & $3	3 reg. operands;
or	or $1,$2,$3	$1 = $2 | $3	3 reg. operands;
xor	xor $1,$2,$3	$1 = $2 □□$3	3 reg. operands;
nor	nor $1,$2,$3	$1 = ~($2 |$3)	3 reg. operands;
and immediate	andi $1,$2,10	$1 = $2 & 10	Logical AND reg, constant
or immediate	ori $1,$2,10	$1 = $2 | 10	Logical OR reg, constant
xor immediate	xori $1, $2,10	$1 = ~$2 &~10	Logical XOR reg, constant

MIPS data transfer instructions

Instruction	*Comment*
LW $1, 30($2)	Load word
LH $1, 40($3)	Load half a word
LB $1, 40($3)	Load byte
SW $3, 500($4)	Store word
SH $3, 502($2)	Store half
SB $2, 41($3)	Store byte

In MIPS64: LD (load double word)

LD $1, 30($2) load a double-word

SD $3, 500($4) store a double-word

Compare and Branch

° Compare and Branch
- BEQ rs, rt, offset   if R[rs] == R[rt] then PC-relative branch
- BNE rs, rt, offset   <>

° Compare to zero and branch
- BLEZ rs, offset   if R[rs] <= 0 then PC-relative branch
- BGTZ rs, offset   >
- BLT   <
- BGEZ   >=
- BLTZAL rs, offset   if R[rs] < 0 then branch and link (into R 31)
- BGEZAL   >=

Reading and Preview

- **Reading:**

CO 4: Chapter 2 (MIPS)

CA 5: Appendix A (ISA)