ECE4680
Computer Organization & Architecture

MIPS Instruction Set Architecture

Why is MIPS a good example?

Learn ISA further by this example.

How does IS fill up the gap between HLL and machine?

RISC Vs. CISC

RISC→CISC

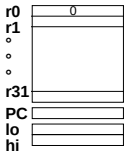
- Determined by VLSI technology.
- Software cost goes up constantly. To be convenient for programmers.
- To shorten the semantic gap between HLL and architecture without advanced compilers.
- To reduce the program length because memory was expensive.
- VAX 11/780 reached the climax with >300 instructions and >20 addressing modes.

CISC→RISC

- Things changed: HLL, Advanced Compiler, Memory size, …
- Finding: 25% instructions used in 95% time.
- Size: usually <100 instructions and <5 addressing modes.
- Other properties: fixed instruction format, register based, hardware control…
- Gains: CPI is smaller, Clock cycle shorter, Hardware simpler, Pipeline easier
- Loss: Program becomes longer, but memory becomes larger and larger, cheaper and cheaper. Programmability becomes poor, but people use HLL instead of IS.
- Result: although program is prolonged, the total gain is still a plus.

MIPS R2000 / R3000 Registers

- 32-bit machine --> Programmable storage 2^{32} x bytes
- 31 x 32-bit GPRs (R0 = 0)
- 32 x 32-bit FP regs (f0 - f31, paired DP)
- HI, LO, PC: SPRegisters
- Big Endian
- 2 nomenclatures(next slide)
- See Fig. A.18 at P. A-50 for more details



- Addressing modes:
  - immediate
  - register
  - displacement
- All instructions are 32-bit wide and must be aligned.

2 Nomenclatures of MIPS Registers (p.140, A-23)

Name	number	Usage	Reserved on call?
zero	0	constant value =0	n.a.
at	1	reserved for assembler (p.147,157)	n.a.
v0 – v1	2 – 3	values for results and expression evaluation	no
a0 – a3	4 – 7	arguments	no
t0 – t7	8 – 15	temporaries	no
s0 – s7	16 – 23	saved	yes
t8 – t9	24 – 25	more temporaries	no
k0 – k1	26 – 27	Reserved for kernel	n.a.
gp	28	global pointer	yes
sp	29	stack pointer	yes
fp	30	frame pointer	yes
ra	31	return address	yes

zero	at	v0-v1	a0 - a3	t0 - t7	s0 - s7	t8 - t9	k0 - k1	gp	sp	fp	ra
0	1	2 - 3	4 - 7	8 --- 15	16 --- 23	24 - 25	26 - 27	28	29	30	31

MIPS arithmetic and logic instructions

Instruction	*Example*	*Meaning*	*Comments*
add	add $1,$2,$3	$1 = $2 + $3	3 operands; exception possible
subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands; exception possible
add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant; exception possible
multiply	mult $2,$3	Hi, Lo = $2 x $3	64-bit signed product
divide	div $2,$3	Lo = $2 ÷ $3,	Lo = quotient, Hi = remainder
			Hi = $2 mod $3
Move from Hi	mfhi $1	$1=Hi	get a copy of Hi
Move from Lo	mflo $1	$1=lo	

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

Example (p110)

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
add $8, $4, $5
sub $1, $7, $8
```

MIPS data transfer instructions

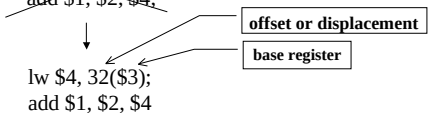
Instruction	*Comment*
SW 500($4), $3	Store word
SH 502($2), $3	Store half
SB 41($3), $2	Store byte
LW $1, 30($2)	Load word
LH $1, 40($3)	Load half a word
LB $1, 40($3)	Load byte

Example (pp112-114)

Assume A is an array of 100 words, and compiler has associated the varialbes 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);
add $1, $2, $4

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

Example (p114)

Assume A is an array of 100 words, and compiler has associated the varialbes 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]

```
addi  $6, $0, 4;      $6 = 4
mult  $5, $6;         Hi,Lo = i*4
mflo  $6;             $6 = i*4, assuming i is small

add $4, $3, $6;       $4 ← address of  A[i]

add $1, $2, $4
```


MIPS jump, branch, 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	
Pseudoinstruction	*blt, ble, bgt, bge*	*not implemented by hardware, but synthesized by assembler*
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.	
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	

Example (p123)

```
        if (i==j) go to L1;  ——————→  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  —pseudoinstruction→  bqe $4, $5, L1
        add $1, $2, $3                              ↓ assembler
L1:     sub $1, $1, $4                       slt  $1, $4, $5
                                             beq $0, $1, L1
```


Example (p126)

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

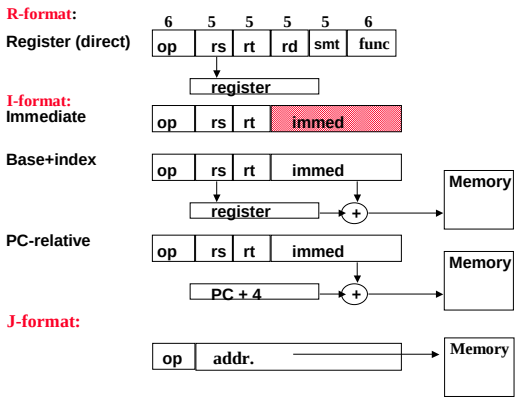
Example (p127)

```
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
       add $17, $17, $18
       j      Loop
Exit:
```


MIPS Addressing Modes/Instruction Formats (p118,148,152)



Example: See machine code in memory (p149)

```
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

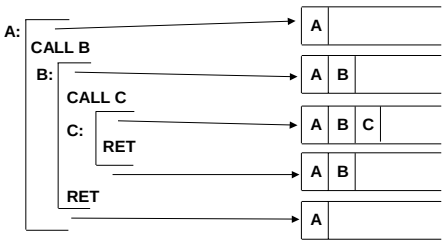
```
Loop:  add $20, $17, $17
       add $20, $20, $20
       add $20, $20, $3
       lw $21,0($20)
       bne $21, $19, Exit
       add $17, $17, $18
       j      Loop
Exit:
```

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

- Offset in branch is relative. Address in jump is absolute.
- Address in Branch or Jump instruction is word address so that they can go 4 times far as opposed to byte address. (p150)

Procedure Call and Stack

Stacking of Subroutine Calls & Returns and Environments:



Some machines provide a memory stack(special hardware) as part of the architecture (e.g. the VAX). Use special instructions, e.g. pop, push.

Sometimes stacks are implemented via software convention (e.g. MIPS). Use same data transfer instructions, e.g., lw, sw.

Example in C: swap (pp163-165)

- Assume swap is called as a procedure
- Assume temp is register $15; arguments v and k ~ $16 and $17;
- Write MIPS code

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

```
sll   $18, $17, 2     ; mulitply k by 4
addu  $18, $18, $16   ; address of v[k]
lw    $15, 0($18)     ; load v[k]
lw    $19, 4($18)     ; load v[k+1]
sw    $19, 0($18)     ; store v[k+1] into v[k]
sw    $15, 4($18)     ; store old v[k] into v[k+1]
```

Registers $15, $16, $17, $18, $19 are occupied by caller ??

Example: Swap

Given a procedure swap(v, j)

Calling swap is as simple as

```
jal swap
```

jal --- jump and link

```
$31 = PC+4;      $31=$ra : always store return address
goto swap
```

swap: MIPS

```
swap:
   addi   $sp,$sp, –24      ; Make room on stack for 6 registers
   sw     $31, 20($sp)      ; Save return address
   sw     $15, 16($sp)      ; Save registers on stack
   sw     $16, 12($sp)
   sw     $17, 8($sp)
   sw     $18, 4($sp)
   sw     $19, 0(sp)
   ....
   lw     $19, 0($sp)       ; Restored registers from stack
   lw     $18, 4($sp)
   lw     $17, 8($sp)
   lw     $16, 12($sp)
   lw     $15, 16($sp)
   lw     $31, 20($sp)      ; Restore return address
   addi   $sp,$sp, 24       ; restore top of stack
   jr     $31               ; return to place that called swap
```


Other ISAs

- Intel 8086/88 => 80286 => 80386 => 80486 => Pentium => P6
  - 8086 few transistors to implement 16-bit microprocessor
  - tried to be somewhat compatible with 8-bit microprocessor 8080
  - successors added features which were missing from 8086 over next 15 years
  - product of several different Intel engineers over 10 to 15 years
  - Announced 1978
- VAX simple compilers & small code size =>
  - efficient instruction encoding
  - powerful addressing modes
  - powerful instructions
  - few registers
  - product of a single talented architect
  - Announced 1977

Machine Examples: Address & Registers

Intel 8086	2^{20} x 8 bit bytes AX, BX, CX, DX SP, BP, SI, DI CS, SS, DS IP, Flags	acc, index, count, quot stack, string code,stack,data segment
VAX 11	2^{32} x 8 bit bytes 16 x 32 bit GPRs	r15-- program counter r14-- stack pointer r13-- frame pointer r12-- argument ptr
MC 68000	2^{24} x 8 bit bytes 8 x 32 bit GPRs 7 x 32 bit addr reg 1 x 32 bit SP 1 x 32 bit PC	
MIPS	2^{32} x 8 bit bytes 32 x 32 bit GPRs 32 x 32 bit FPRs HI, LO, PC	

Homework 2, due Feb. 20, 2002

- Questions 3.2, 3.3, 3.5, 3.6, 3.7, 3.9, 3.11
- To answer question 3.7, please refer to Figure 3.13 (page 140) for register convention
- To answer 3.11, please refer to sort example in pages 166 for a skeleton of for loop