ECE4680  Lec4  MIPS.1
February 6, 2002

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
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.

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
- 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
- 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.

MIPS arithmetic and logic instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add $1,$2,$3</td>
<td>$1 = $2 + $3</td>
<td>3 operands; exception possible</td>
</tr>
<tr>
<td>subtract</td>
<td>sub $1,$2,$3</td>
<td>$1 = $2 – $3</td>
<td>3 operands; exception possible</td>
</tr>
<tr>
<td>add immediate</td>
<td>.add $1,$2,100</td>
<td>$1 = $2 + 100</td>
<td>+ constant; exception possible</td>
</tr>
<tr>
<td>multiply</td>
<td>mul $2,$3</td>
<td>HI, Lo = $2 x $3</td>
<td>64-bit signed product</td>
</tr>
<tr>
<td>divide</td>
<td>div $2,$3</td>
<td>Lo = $2 ÷ $3, Hi = remainder</td>
<td>Hi = $2 mod $3</td>
</tr>
<tr>
<td>Move from Hi</td>
<td>mhfi $1</td>
<td>$1=Hi</td>
<td>get a copy of Hi</td>
</tr>
<tr>
<td>Move from Lo</td>
<td>mlfi $1</td>
<td>$1=lo</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>and $1,$2,$3</td>
<td>$1 = $2 &amp; $3</td>
<td>Logical AND</td>
</tr>
<tr>
<td>or</td>
<td>or $1,$2,$3</td>
<td>$1 = $2</td>
<td>Logical OR</td>
</tr>
<tr>
<td>xor</td>
<td>xor $1,$2,$3</td>
<td>$1 = $2</td>
<td>Logical XOR</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>number</th>
<th>Usage</th>
<th>Reserved on call?</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>0</td>
<td>constant value</td>
<td>n.a.</td>
</tr>
<tr>
<td>at</td>
<td>25</td>
<td>reserved for assembler (p.147,157)</td>
<td>n.a.</td>
</tr>
<tr>
<td>a0 - a3</td>
<td>4 - 7</td>
<td>arguments</td>
<td>no</td>
</tr>
<tr>
<td>t0 - t7</td>
<td>8 - 15</td>
<td>temporaries</td>
<td>no</td>
</tr>
<tr>
<td>t8 - t15</td>
<td>16 - 23</td>
<td>saved</td>
<td>yes</td>
</tr>
<tr>
<td>gp</td>
<td>28</td>
<td>global pointer</td>
<td>yes</td>
</tr>
<tr>
<td>sp</td>
<td>29</td>
<td>stack pointer</td>
<td>yes</td>
</tr>
<tr>
<td>fp</td>
<td>30</td>
<td>frame pointer</td>
<td>yes</td>
</tr>
<tr>
<td>ra</td>
<td>31</td>
<td>return address</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW 500($4), $3</td>
<td>Store word</td>
</tr>
<tr>
<td>SH 502($2), $3</td>
<td>Store half</td>
</tr>
<tr>
<td>SB 41($3), $2</td>
<td>Store byte</td>
</tr>
<tr>
<td>LW $1, 30($2)</td>
<td>Load word</td>
</tr>
<tr>
<td>LH $1, 40($3)</td>
<td>Load half a word</td>
</tr>
<tr>
<td>LB $1, 40($3)</td>
<td>Load byte</td>
</tr>
</tbody>
</table>

Example (pp112-114)

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);   \quad \text{\$4 <-- A[8]}
add $1, $2, $4;
lw $4, 32($3);
add $1, $2, $4
```


```
SW $1, 48($3)
```

Example (p114)

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
\[ g = h + A[i] \]

```
addi $6, $0, 4; \quad \$6 = 4
mult $5, $6; \quad \text{Hi,Lo = i*4}
mflo $6; \quad \text{\$6 = i*4, assuming i is small}
add $4, $3, $6; \quad \$4 <-- address of A[i]
add $1, $2, $4
```

MIPS jump, branch, compare instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch on equal</td>
<td>bne $1,$2,100</td>
<td>if ($1 != $2) go to PC+4+100</td>
</tr>
<tr>
<td>branch on not eq.</td>
<td>bne $1,$2,100</td>
<td>Not equal test; PC relative</td>
</tr>
<tr>
<td>Pseudoinstruction</td>
<td>blt, ble, bgt, bge</td>
<td>Not implemented by hardware, but synthesized by assembler</td>
</tr>
<tr>
<td>set on less than</td>
<td>sll $1,$2,$3</td>
<td>if ($2 &lt; $3) $1=1; else $1=0</td>
</tr>
<tr>
<td>set less than imm.</td>
<td>sll $1,$2,100</td>
<td>Compare less than; 2's comp</td>
</tr>
<tr>
<td>Jump register</td>
<td>j 10000</td>
<td>Jump to target address</td>
</tr>
<tr>
<td>Jump and link</td>
<td>jal 10000</td>
<td>For procedure call</td>
</tr>
</tbody>
</table>

Example (p123)

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; \quad \text{pseudo}\text{instruction} \quad \text{bge $4, $5, L1}\quad \text{assembler}
add $1, $2, $3
L1: \quad \text{sub} $1, $1, $4
```

Example (p128)

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: \quad \text{add} \quad \text{\$7, \$3, \$3; \quad i*2}
\quad \text{add} \quad \text{\$7, \$7, \$7; \quad i*4}
\quad \text{lw} \quad \text{\$6, 0($7); \quad S6=A[i]}
\quad \text{add} \quad \text{\$1, \$1, \$6; \quad g = g+A[i]}
\quad \text{add} \quad \text{\$3, \$3, \$4}
\quad \text{bne} \quad \text{\$3, \$2, Loop;}
```

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: \quad \text{add} \quad \text{\$7, \$3, \$3; \quad i*2}
\quad \text{add} \quad \text{\$7, \$7, \$7; \quad i*4}
\quad \text{lw} \quad \text{\$6, 0($7); \quad S6=A[i]}
\quad \text{add} \quad \text{\$1, \$1, \$6; \quad g = g+A[i]}
\quad \text{add} \quad \text{\$3, \$3, \$4}
\quad \text{bne} \quad \text{\$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:

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:

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];
    sl $18, $17, 2 ; multiply k by 4
    v[k+1] = temp;
    addu $18, $18, $16 ; address of v[k]
}

sw $15, $18($18) ; load v[k]
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 ??
### swap: MIPS

```assembly
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)

sw $31, 20($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

<table>
<thead>
<tr>
<th>Processor</th>
<th>8 bit bytes</th>
<th>16 bit bytes</th>
<th>32 bit bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel 8086</td>
<td>20</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>VAX 11</td>
<td>32</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>MC 68000</td>
<td>24</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>MIPS</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>

- acc, index, count, quot
- stack, string
- code, stack, data segment
- IP, Flags

- r15—program counter
- r14—stack pointer
- r13—frame pointer
- r12—argument ptr

### 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