EECS 322
Computer Architecture

Language of the Machine

Control Flow

Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Case Western Reserve University

This presentation uses powerpoint animation: please viewshow
Memory Organization: byte addressing

- "Byte addressing" means that the index points to a byte of memory.
- C/C++ use byte addressing independent of the memory organization.
- MIPS uses byte addressing although the memory is organized by words (4 bytes).

- Accessing a word on a non-word byte address (unaligned) causes the memory access time to double.
Review: Basic Machine Instructions

**Arithmetic instructions**

- **MIPS syntax:** \( \text{add} \; \$rd, \; \$rs, \; \$rt \)
- **MIPS semantics:** \( \text{reg}[$rd] = \text{reg}[$rs] + \text{reg}[$rt] \)

- **MIPS syntax:** \( \text{addi} \; \$rt, \; \$rs, \; \text{imm16} \)
- **MIPS semantics:** \( \text{reg}[$rt] = \text{reg}[$rs] + \text{imm16} \)

- **MIPS syntax:** \( \text{sub} \; \$rd, \; \$rs, \; \$rt \)
- **MIPS semantics:** \( \text{reg}[$rd] = \text{reg}[$rs] - \text{reg}[$rt] \)

- **MIPS syntax:** \( \text{subi} \; \$rt, \; \$rs, \; \text{imm16} \)
- **MIPS semantics:** \( \text{reg}[$rt] = \text{reg}[$rs] - \text{imm16} \)
Data Transfer Machine Instructions: lw, lbu

- **MIPS syntax:** lw $rt, offset16($rs)

- **semantics:** \( \text{reg}[\text{rt}] = \text{mem}[/\text{offset} + \text{reg}[\text{rs}]/]\)

- **C language:** int mem[2^{30}];
  
  \$rt = \text{mem}[(\text{offset} + \text{rs})/4];

- **MIPS syntax:** lbu $rt, offset16($rs)

- **semantics:** \( \text{mem}[/\text{offset} + \text{reg}[\text{rs}]/] = \text{reg}[\text{rt}]/\)

- **C language:** unsigned char mem[2^{32}];
  
  \$rt = \text{mem}[/\text{offset} + \text{rs}];
  
  \$rt = \text{rt} & 0x000000ff;
Data Transfer Machine Instructions: lui

- Load upper immediate

  Loads constant with 16 bits

- **MIPS syntax:** \texttt{lui $rt, imm16}

- **semantics:** \texttt{reg[$rt][31..16]=imm16;}

- **C language:**
  
  \begin{verbatim}
  int mem[2^30];
  $rt = (imm16<<16) | (0x0000ffff) & $rt;
  \end{verbatim}
How about larger constants?

- We'd like to be able to load a 32 bit constant into a register
- Must use two instructions, new "load upper immediate" instruction

lui $t0, 1010101010101010

Then must get the lower order bits right, i.e.,
ori $t0, $t0, 1010101010101010
Data Transfer Machine Instructions: sw, sb

- **MIPS syntax:** sw $rt, offset16($rs)

- **semantics:** \( \text{mem}[\text{offset + reg}[$rs]] = \text{reg}[$rt] \)

- **C language:**
  \[
  \text{int mem}[2^{30}];
  \text{mem}[(\text{offset + } \text{$rs})/4] = \text{$rt};
  \]

- **MIPS syntax:** sb $rt, offset16($rs)

- **semantics:** \( \text{mem}[\text{offset + reg}[$rs]] = \text{reg}[$rt] \)

- **C language:**
  \[
  \text{unsigned char mem}[2^{32}];
  \text{mem}[(\text{offset + } \text{$rs})] = \text{$rs} \& 0x000000ff;
  \]
Logical Machine Instructions

- bitwise and: \texttt{and} $rd$, $rs$, $rt$
  - C language: \texttt{$rd = $rs \& $rt$; /* 1\&1=1; 0\&X=0 */}
- bitwise or: \texttt{or} $rd$, $rs$, $rt$
  - C language: \texttt{$rd = $rs \mid $rt$; /* 1\mid X=1; 0\mid X=0 */}
- bitwise or: \texttt{ori} $rt$, $rs$, imm16
  - C language: \texttt{$rd = $rs \mid imm16;}
- bitwise or: \texttt{andi} $rt$, $rs$, imm16
  - C language: \texttt{$rd = $rs \& imm16;}

- shift left: \texttt{sll} $rt$, $rs$, imm16
  - C language: \texttt{$rt = $rs \ll imm16;}
- unsigned right: \texttt{srl} $rt$, $rs$, imm16
  - C language: \texttt{$rt = $rs \gg imm16;}

CWRU EECS 322  8
Unsigned char Array example

int Array:

```c
register int g, h, i;
int A[66];
g = h + A[i];
```

Compiled MIPS:

```mips
add $t1,$s4,$s4  # $t1 = 2*i
add $t1,$t1,$t1  # $t1 = 4*i
add $t1,$t1,$s3  #$t1=& A[i]
lw $t0,0($t1)  # $t0 = A[i]
add $s1,$s2,$t0  # g = h + A[i]
```

unsigned char Array:

```c
register int g, h, i;
unsigned char A[66];
g = h + A[i];
```

Load byte unsigned:
load a byte and fills the upper 24 register bits with zeros.

```mips
add $t1,$t1,$s4
lbu $t0,0($t1)
add $s1,$s2,$t0
```
if/else conditional statements

• if statements in C
  • if (condition) statement1
  • if (condition) statement1 else statement2

• Following code is
  
  if (condition) goto L1;
  statement2;
  goto L2;

  L1: statement1;
  L2:

• Actual implementations

  if (! condition) goto L1;
  statement1;
  goto L2;

  L1: statement2;
  L2:

C/C++ does have a goto keyword

! is logical not in C/C++
beq/bne: conditional branches

- Decision instruction in MIPS:
  \[
  \text{beq register1, register2, L1}
  \]
  beq is “Branch if (registers are) equal”

- Same meaning as (using C):
  \[
  \text{if (register1 == register2) goto L1;}
  \]

- Complementary MIPS decision instruction
  \[
  \text{bne register1, register2, L1}
  \]
  bne is “Branch if (registers are) not equal”

- Same meaning as (using C):
  \[
  \text{if (register1 != register2) goto L1;}
  \]

Most common C/C++ mistake

\[
\text{== comparison = assignment}
\]
Conditional example

- C code fragment
  if (i == j) { f=g+h; }
  else { f=g-h; }

- re-written C code
  if (i != j) goto L1;
      f=g+h;
  goto L2;
L1:
      f=g-h;
L2:

- MIPS code
  bne $s3,$s4,L1
  add $s0,$s1,$s2
  j L2
L1:
  sub $s0,$s1,$s2
L2:
if conditional

- The condition is compared with zero or not zero

- For example
  \[
  \text{if (i) \{ f=g+h; \}}
  \]
  is the same as
  \[
  \text{if (i \neq 0) \{ f=g+h; \}}
  \]

- Another example
  \[
  \text{if (! i) \{ f=g+h; \}}
  \]
  is the same as
  \[
  \text{if (i == 0) \{ f=g+h; \}}
  \]

- This allows the $0$ register to be exploited by C/C++

- Again, another reason, C is successful
The ?: conditional expression

- conditional statements in C
  - variable = condition ? statement1 : statement2

- Example: f = (i == j) ? g+h : g-h;

- Following code is
  
  if (! condition) goto L1;
  variable=statement1;
  goto L2;

  L1:
  variable=statement2;

  L2:

  MIPS code example
  
  bne $s3,$s4,L1
  add $s0,$s1,$s2
  j L2

  L1:
  sub $s0,$s1,$s2

  L2:
Control flow: loops - while

• while statements in C
  
  • while (condition) statement1

• Review if/else
  
  if (! condition) goto L1;
  
  statement1;
  goto L2;

  L1: statement2;
  
  L2:

• observe: while L2 is the same as a conditional if that now loops back on itself.

• observe: while L1 is the same as a conditional else

• while loop implementation
  
  L2: if (! condition) goto L1;

  statement1;
  goto L2;

  L1:
Control flow: loops - for

- for statements in C
  - for (initialize; condition; increment) statement1

is equivalent to
initialize;
while (condition) {
  statement1;
  increment;
}

• Actual implementation
initialize;
L2: if (! condition) goto L1;
  statement1;
  increment;
  goto L2;
L1:
slt: Set on Less Than

- So far $==$, $!=$; what about $<$ or $>$?
- MIPS instruction “Set Less Than”
  
  \[
  \text{slt} \quad \text{rd}, \text{rs}, \text{rt}
  \]

- Meaning of slt in C
  
  \[
  \text{if} \ (\text{rs} < \text{rt}) \ \{ \ \text{rd} = 1; \ \} \ \text{else} \ \{ \ \text{rd} = 0; \ \}
  \]

- Alternately
  
  \[
  \text{rd} = (\text{rs} < \text{rt}) \ ? \ 1 : 0;
  \]

- Then use branch instructions to test result in \(\text{rd}\)

- Other variations
  
  \[
  \begin{align*}
  & \text{slti} \ \text{rd}, \text{rs}, \text{immed16} \quad \# \ \text{rd}=(\text{rs} < \text{immed16})?1:0; \\
  & \text{sltu} \ \text{rd}, \text{rs}, \text{rt} \quad \# \ \text{unsigned int} \\
  & \text{sltiu} \ \text{rd}, \text{rs}, \text{immed16} \quad \# \ \text{unsigned int}
  \end{align*}
  \]
slt example

- C code fragment
  
  ```c
  if (i < j) { f=g+h; }
  else      { f=g-h; }
  ```

- re-written C code
  
  ```c
  temp = (i < j)? 1 : 0;
  if (temp == 0) goto L1;
  
  f=g+h;
  goto L2;
  
  L1:
  f=g-h;
  
  L2:
  ```

- MIPS code
  
  ```mips
  slt $t1,$s3,$s4
  beq $t1,$0,L1
  add $s0,$s1,$s2
  jL 2
  
  L1:
  sub $s0,$s1,$s2
  
  L2:
  ```

The $0 register becomes useful again for the beq
Control flow: switch statement

- Choose among four alternatives depending on whether k has the value 0, 1, 2, or 3

```c
switch (k) {
    case 0: f=i+j; break; /* k=0*/
    case 1: f=g+h; break; /* k=1*/
    case 2: f=g–h; break; /* k=2*/
    case 3: f=i–j; break; /* k=3*/
}
```

- Could be done like chain of if-else

```c
if ( k == 0 ) f=i+j;
else if ( k == 1 ) f=g+h;
    else if ( k == 2 ) f=g–h;
        else if ( k==3 ) f=i–j;
```

The switch case is restricted to constant expressions. This is to intentional in order to exploit the hardware.
Switch MIPS example: subtraction chain

- Could be done like chain of if-else if(k==0) f=i+j;
  else if(k==1) f=g+h;
  else if(k==2) f=g–h;
  else if(k==3) f=i–j;

```
bne $s5,$zero, L1  # branch k!=0
add $s0,$s3,$s4    #k==0 so f=i+j
j Exit
L1: subi $t0,$s5,1  # $t0=k-1
    bne $t0,$zero,L2 # branch k!=1
    add $s0,$s1,$s2   #k==1 so f=g+h
    j Exit
L2: subi $t0,$s5,2  # $t0=k-2
    bne $t0,$zero,L3 # branch k!=2
    sub $s0,$s1,$s2   #k==2 so f=g-h
    j Exit
L3: sub  $s0,$s3,$s4 #k==3 so f=i-j
    Exit:
```
signed char Array example

unsigned char Array:
    register int g, h, i;
    unsigned char A[66];
    g = h + A[i];

Load byte unsigned:
load a byte and fills the upper 24 register bits with zeros.

if ($t0 >= 128) { $t0 |= 0xffffffff; }

add $t1,$t1,$s4
lbu $t0,0($t1)
add $s1,$s2,$t0

Data types make a big impact on performance!

8 bit sign = 128 = 0x00000080

add $t1,$t1,$s4
lbu $t0,0($t1)
slti $t1,$t0,128
bne $t1,$0,L2
ori $t0,0xff00
lui $t0,0xffff
L2: add $s1,$s2,$t0

signed char Array:
    register int g, h, i;
    signed char A[66];
    g = h + A[i];
Class Homework: due next class

Chapter 3 exercises (pages 197-):

3.1, 3.4, 3.6 (by hand only), 3.11