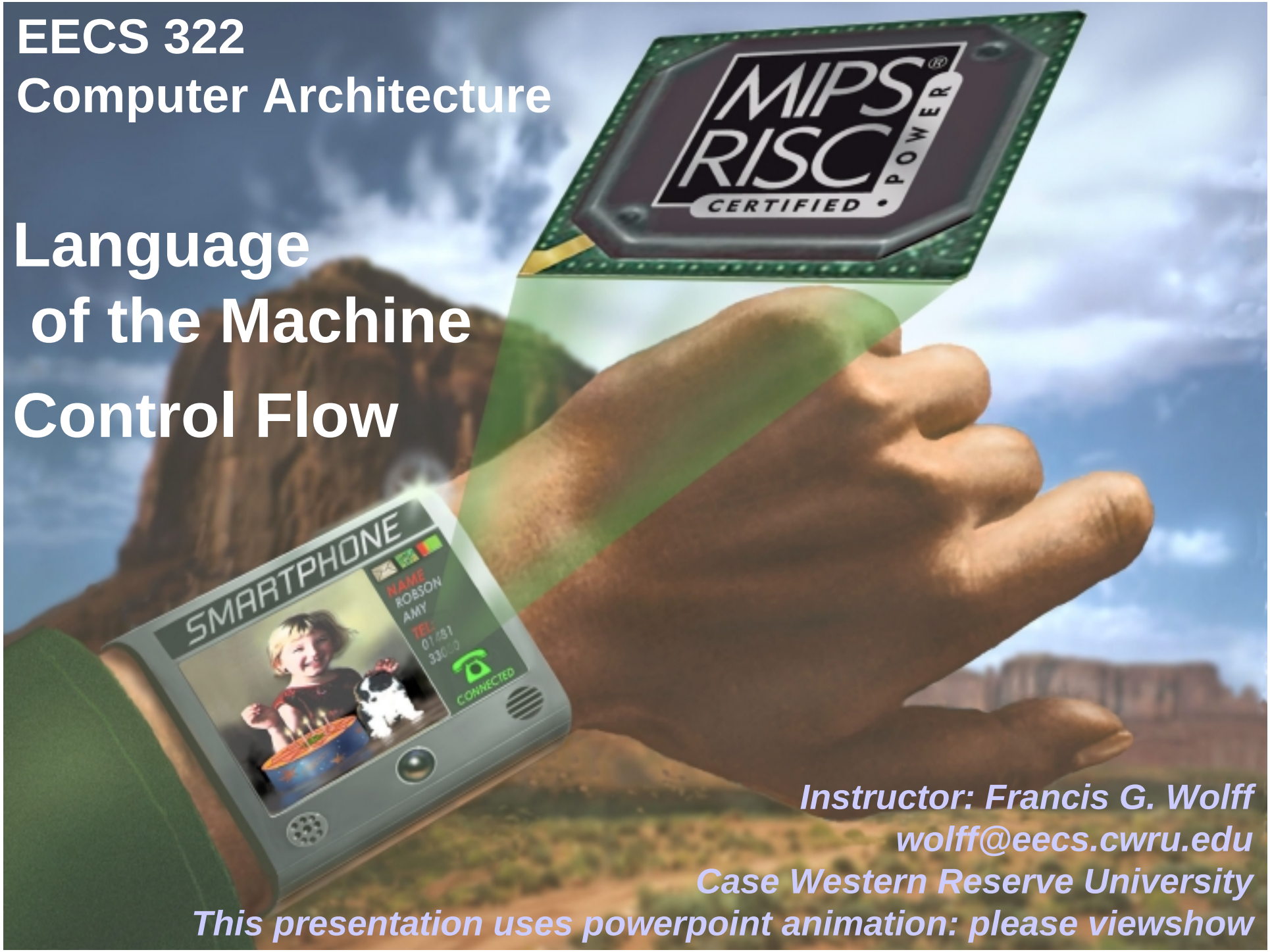
EECS 322
Computer Architecture



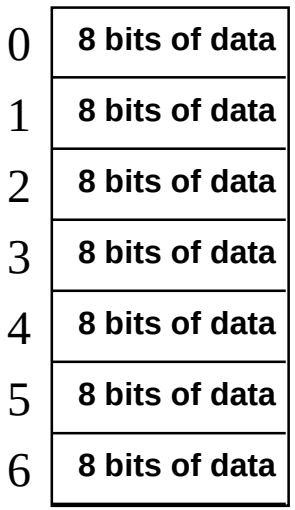
Language of the Machine
Control Flow

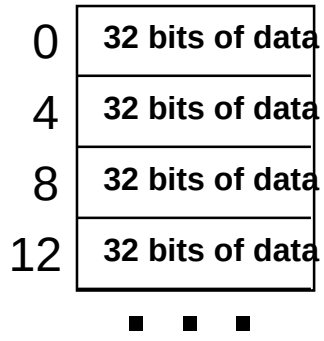
Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Case Western Reserve University

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*:** **add $rd, $rs, $rt**
- **MIPS *semantics*:** reg[$rd] = reg[$rs] + reg[$rt]

- **MIPS *syntax*:** **addi $rt, $rs, imm16**
- **MIPS *semantics*:** reg[$rt] = reg[$rs] + imm16

- **MIPS *syntax*:** **sub $rd, $rs, $rt**
- **MIPS *semantics*:** reg[$rd] = reg[$rs] - reg[$rt]

- **MIPS *syntax*:** **subi $rt, $rs, imm16**
- **MIPS *semantics*:** reg[$rt] = reg[$rs] - imm16

Data Transfer Machine Instructions: lw, lbu

- **MIPS *syntax*:** **lw $rt, offset16($rs)**
- ***semantics*:** reg[$rt] = mem[offset + reg[$rs]]
- **C language:** int mem[2^{30}];
  $rt = mem[(offset + $rs)/4];

- **MIPS *syntax*:** **lbu $rt, offset16($rs)**
- ***semantics*:** mem[offset + reg[$rs]] = reg[$rt]
- **C language:** unsigned char mem[2^{32}];
  $rt = mem[offset + $rs] ;
  $rt = $rt & 0x000000ff;

Data Transfer Machine Instructions: lui

- **Load upper immediate**

  Loads constant with 16 bits

- **MIPS *syntax*:** **lui $rt, imm16**
- ***semantics*:** reg[$rt][31..16]=imm16;
- **C language:** int mem[2^{30}];

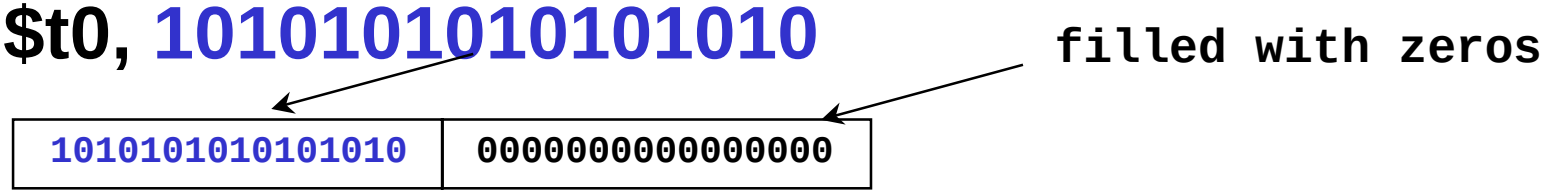
  $rt = (imm16<<16) | (0x0000ffff) & $rt;

How about larger constants?

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

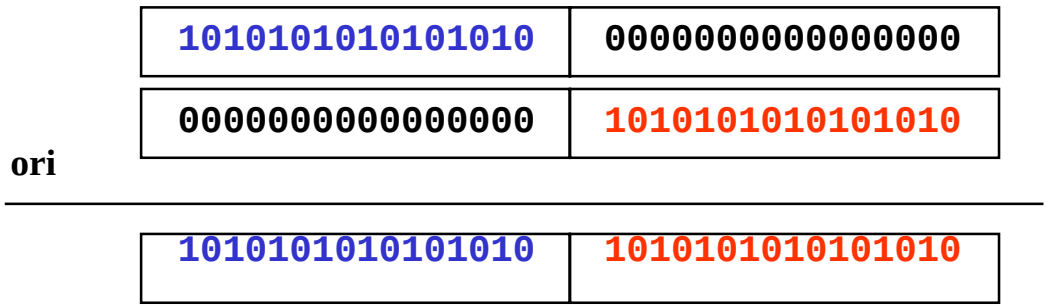
  new "load upper immediate" instruction

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

- **MIPS *syntax*:** **sb $rt, offset16($rs)**
- ***semantics*:** mem[offset + reg[$rs]] = reg[$rt]
- **C language:** unsigned char mem[2^{32}];
  mem[offset + $rs] = $rs & 0x000000ff;

Logical Machine Instructions

- **bitwise and:** **and $rd, $rs, $rt**
- **C language:** $rd = $rs & $rt; /* 1&1=1; 0&X=0 */
- **bitwise or:** **andi $rt, $rs, imm16**
- **C language:** $rd = $rs & imm16;

- **bitwise or:** **or $rd, $rs, $rt**
- **C language:** $rd = $rs | $rt; /* 1|X=1; 0|X=0 */
- **bitwise or:** **ori $rt, $rs, imm16**
- **C language:** $rd = $rs | imm16;

- **shift left:** **sll $rt, $rs, imm16**
- **C language:** $rt = $rs << imm16;

- **unsigned right:** **srl $rt, $rs, imm16**
- **C language:** $rt = $rs >> imm16;

Unsigned char Array example

int Array:

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

Compiled 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:

```
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.

```
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:**

  beq register1, register2, L1

  beq is “Branch if (registers are) equal”

- **Same meaning as (using C):**

  if (register1 == register2) goto L1;

Most common C/C++ mistake
== comparison = assignment

- **Complementary MIPS decision instruction**

  bne register1, register2, L1

  bne is “Branch if (registers are) not equal”

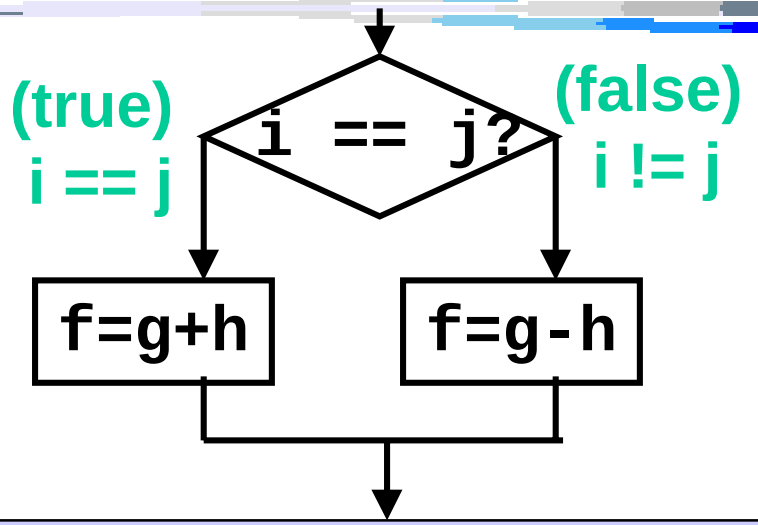
- **Same meaning as (using C):**

  if (register1 != register2) goto L1;

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

```
if (i) { f=g+h; }
```

  is the same as

```
if (i != 0) { f=g+h; }
```

  - Another example

```
if (! i) { f=g+h; }
```

  is the same as

```
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:
```

- **while loop implementation**

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

- **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**

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”

```
slt $rd,$rs,$rt
```

- Meaning of slt in C

```
if ($rs < $rt) { $rd = 1; } else { $rd = 0; }
```

- Alternately

```
$rd = ($rs < $rt) ? 1 : 0;
```

- Then use branch instructions to test result in $rd
- Other variations

```
slti $rd,$rs,immed16    # $rd=($rs < immed16)?1:0;
sltu $rd,$rs,$rt        # unsigned int
sltiu $rd,$rs,immed16   # unsigned int
```

slt example

- C code fragment

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

The $0 register becomes useful again for the beq

- re-written C code

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

- MIPS code

```
        slt   $t1,$s3,$s4
        beq   $t1,$0,L1
        add   $s0,$s1,$s2
        j     L2
L1:
        sub   $s0,$s1,$s2
L2:
```

Control flow: switch statement

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

```
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*/
}
```

The **switch case** is restricted to **constant expressions.** This is to intentional in order to exploit the hardware.

- **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;
```

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              # end of case so 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              # end of case so 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              # end of case so 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 |=  0xffffff00; }

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

signed char Array:

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

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
```

- Data types make a big impact on performance!

Class Homework: due next class

Chapter 3 exercises (pages197-):

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