CS 2630
Computer Organization

Meeting 3: bits, and MIPS intro

Brandon Myers
University of Iowa
Where we are going

Instruction set architecture (e.g., MIPS)

Compiler
- translating source code (C or Java)
- Programs to assembly language
- And linking your code to
- Library code

How the software talks
To the hardware

Memory system

Processor

I/O system

Datapath & Control

Digital logic

How a processor runs MIPS
Programs!

How switches (1 or 0) can be used
to build Interesting functions:
from integer arithmetic to
programmable computers
Where we are going

Instruction memory

Execution engine

Data memory

Learn what gets stored in here
Organize bytes into machine words

(note address in hex)

32-bit words

64-bit words
Peer instruction:

We have a hypothetical word-addressed machine with 32-bit addresses. Assume array `x` starts at address 0x50.

```java
int[] x = new int[4];
```

What is the address of `x[3]`?

a) 0x50  
b) 0x53  
c) 0x5C  
d) 0x60  
e) 0x54
## Review of bitwise operations

<table>
<thead>
<tr>
<th>Logical AND (&amp;)</th>
<th>Logical OR (|)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>0011</td>
</tr>
<tr>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td><strong>0010</strong></td>
<td><strong>1011</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XOR (^)</th>
<th>negate (~)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>0110</td>
</tr>
<tr>
<td>1010</td>
<td>0110</td>
</tr>
<tr>
<td><strong>1001</strong></td>
<td><strong>1001</strong></td>
</tr>
</tbody>
</table>
Shift

• shift: move bits left or right

• Left shift: 6 << 2

• Right shift: 6 >> 1

(diagrams show that I am storing the integers using 5 bits)
Peer instruction

What is the integer result from evaluating this expression? (assume integers are 32 bits)

$(7 << 2) \& 15$
Let’s play a card game

• Come up with a binary encoding for a 52-card deck
Operations on a pair of cards

• We want the following operations to be easy to implement
  • Compare two cards, which is higher numeric value?
  • Compare two cards, are they the same suit?
Administrivia:

• ?new DYB time?
• New features in MyUI
  • University administration says update your student record with new information including gender identity, name pronunciation, and preferred name
Where we are going

Instruction set architecture (e.g., MIPS)

Compiler
- translating source code (C or Java)
- Programs to assembly language
- And linking your code to Library code

How the software talks To the hardware

Memory system

Processor

I/O system
- How a processor runs MIPS Programs!

Datapath & Control
- How switches (1 or 0) can be used to build Interesting functions:
  from integer arithmetic to programmable computers

Digital logic
Representations of a program

```
int x = arr[1];
arr[2] = x + 10;
```

High level language program (human readable)

```
lw $t0, 4($r0)
addi $t0, $t0, 10
sw $t0, 8($r0)
```

assembly program as text (CS2630 student readable)

```
1000111000001000000000000000100
00100001000010000000000000001010
10101110000010000000000000001000
```

assembly program as binary (machine readable)
Languages

• High level language (HLL) programs are **machine-independent**
  • language is convenient to use
  • language is has powerful features

• Assembly language programs are **specific to an architecture**
  • the “native language” of the processor
  • language is bare-bones: lacks most of the features of HLLs
Brief history of compatibility

pre IBM 360

- my_program.asm version 1
  Processor model 1
  slow processor, small memory

- my_program.asm version 2
  Processor model 2
  mediocre processor, medium memory

- my_program.asm version 3
  Processor model 3
  fast processor, large memory

post IBM 360

- my_program.asm
  Processor 1
  slow processor, small memory

- Processor 2
  mediocre processor, medium memory

- Processor 3
  fast processor, large memory
Peer instruction

I wrote my program in C (a high level language) and compiled it to run on a **MIPS machine**. What should I do to get that program to run on an **x86 machine**?

a) **rewrite** my C program **to be compatible** with x86
b) **rewrite** my program **in x86** assembly language
c) **recompile** my C program for x86
d) **reassemble** my program using the x86 assembler
Languages

• High level language: usually has variables, objects, and arrays

• Assembly language: usually just has registers and memory
The main idea

• 4 categories of *instructions*
  • perform an operation on two registers and store result in a register
  • perform an operation on one register and a constant and store the result in a register
  • move a value between a register and memory
  • determine which instruction to execute next