CSI333 Lecture 3

Seth Chaiken

September 12, 2000

House Architecture One

Structure

ROOF

FLOORS

WALLS

Function

Provide sheltered living spaces for people.
The same 32 bit computer word can represent:

Integers $i$, $0 \leq i \leq 2^{32} - 1$

Unsigned Binary

4 Letter (character) sequences

8-8-8-8 ASCII characters

Machine Instructions

6-5-5-16 (R) and other formats

Integers $i$, $-2^{31} \leq i \leq 2^{31} - 1$

Signed (2-s complement) Binary
(certain) Fractional numbers: 101.25, $-\frac{3}{8}$, 
$\approx 6.023 \times 10^{23}$

Single precision IEEE 754 Floating Point 1-8-23

See Goodwin and Miller Ch. 6

---

**Fetch-execute cycle**

What the processor does

do {
    (1) Fetch instruction. (from Memory)
    (2) Decode instruction.
    (3) Execute instruction.
} while (instruction is not "halt")
Each (RISC) machine instruction only does **one**

- arithmetic operation or
- test/control operation or
- memory/processor data transfer
  (with address calculation)
  at a time.

### Slide 7

<table>
<thead>
<tr>
<th>C++</th>
<th>MIPS Assembly Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>int X, Y, Z;</td>
<td>lw $t0, Y</td>
</tr>
<tr>
<td>Z = X + Y*Z;</td>
<td>lw $t1, Z</td>
</tr>
<tr>
<td>if( Z &lt; 0 )</td>
<td>mul $t2, $t1, $t0</td>
</tr>
<tr>
<td>{</td>
<td>#compute Y*Z</td>
</tr>
<tr>
<td>Z = 0;</td>
<td>lw $t3, X</td>
</tr>
<tr>
<td>}</td>
<td>add $t4, $t3, $t2</td>
</tr>
<tr>
<td>#compute X + Y*Z into $t4</td>
<td></td>
</tr>
<tr>
<td>LabelOne:</td>
<td>bge $t4, $0, LabelOne</td>
</tr>
<tr>
<td>}</td>
<td>li $t4, 0</td>
</tr>
<tr>
<td>#put 0 in $t4 if( Z &lt; 0 )</td>
<td></td>
</tr>
<tr>
<td>sw $t4, Z #store result</td>
<td></td>
</tr>
</tbody>
</table>
Lesson of Object Oriented Programming:
Understanding the properties, meaning, and types of DATA is MORE IMPORTANT than algorithms and executable programs.

Slide 9

```c
int X, Y, Z;
Z = X + Y*Z;
if( Z < 0 )
{
    Z = 0;
}
```

Variables

<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
</tr>
</tbody>
</table>

store values and
all look the same in C/C++
In **Instruction Set Architecture** (machine language) there are three facilities for “remembering” data:

- Data registers (called **Registers** for short).
- RAM (called **Memory** for short).
- Some I/O devices (disks, etc.)

They have different **performance** (i.e. speed) properties and are programmed differently.

---

**Slide 10**

**Registers**

**Located in Processor**

Identified by number: 0, 1, ..., 31

(also by software convention register names: $\texttt{zero}$, $\texttt{at}$, $\texttt{v0}$-$\texttt{v1}$, $\texttt{a0}$-$\texttt{a3}$, $\texttt{t0}$-$\texttt{t9}$, $\texttt{s0}$-$\texttt{s7}$, $\texttt{k0}$-$\texttt{k1}$, $\texttt{gp}$, $\texttt{sp}$, $\texttt{fp}$, $\texttt{ra}$

$\texttt{0}=$\texttt{zero} and $\texttt{ra}=$\texttt{31} have special hardware behavior.)

- Size of each: 32 bits
- Few in number (31 or 32)
- Total storage space: $31 \times 4 = 124$ bytes
- Access time: 2 Nanoseconds to access 3 registers
- Expensive per bit

**Slide 11**
Memory
Located (mostly) outside the Processor
Names (called addresses): 0, 1, ..., $2^{32} - 1$
Addresses locate byte (8 bit) sized chunks, but usually
one 4 byte word is accessed at a time.
Many in number
Late 90’s PCs have tens of Megabytes to Gigabytes of
RAM
Roughly 20 ns (uncached), or more.
Cheap per bit

How to use
Registers
Code 2 or 3 reg. numbers in one mach. instruction:
\begin{align*}
  \text{add } & \$t0, \$t1, \$t2 \\
  \text{(3 register add.)} \\
  \text{addi } & \$t0, \$t1, 365 \\
  \text{(add immediate constant} \\
  \text{365 with } \$t1, \text{put sum in} \\
  \text{\$t0) }
\end{align*}

memory
Make the processor use registers and immediate constants to compute the Memory Address.
Load and Store instructions move data between Memory and a register.
Interface to the Processor

- **IN**: 32 electric wires, Memory Address
- **IN**: 1 wire, Write/Read (command)
- **IN/OUT**: 32 wires, Data Bits

What C Pointer Values Are!

Memory (RAM)

Values Are!

(addr. translation, caches, handshakes, many other details omitted)
Example 1:
Required computation (C++):

```cpp
int f, g, h, i, j;
f = (g + h) - (i + j);
```

MAL code:

```
# Register Assignments
# f : $15      g : $16
# h : $17      i : $18
# j : $19

# INSTRUCTIONS       # INVARIANTS
add $8, $16, $17   # $8 has g+h
add $9, $18, $19   # $9 has i+j
sub $15, $8, $9    # $15 has f.
```

You MUST document MAL programs this way for MAL programming assignments!!
An invariant is a comment about relationships among data that is ALWAYS TRUE when control flow reaches that comment, if you didn’t make a mistake.

Programming Tip: Convert invariants to C/C++ statements (predicates) and TEST them in the assert() macro.
Example 2: C++ segment:

```cpp
int f, g, h, i, j;
if (i == j)
    f -= i;
else
    f = g + h;
```

MAL segment:

```
# Register assignments like Ex. 1:
# f: $15  g: $16  h: $17
#   i: $18  j: $19
beq $18, $19, ThenL
      # i != j here
add $15, $16, $17  # f = g+h.
j exitL
ThenL:  # i == j here
sub $15, $15, $18  # f -= i.
exitL:
```
Example 3:

C++ segment: (All variables are of type int.)

```cpp
if (a > 0) {
    x += y; p *= q;
}
```

Register assignment:

- `a`: $15
- `x`: $16
- `y`: $17
- `p`: $18
- `q`: $19

Equivalent MAL segment: (Version I)

```MAL
bgt $15, $0, then
j end_if
then: add $16, $16, $17
mul $18, $18, $19
end_if:
```
Equivalent MAL segment: (Version II)

```text
ble  $15, $0, end_if
add  $16, $16, $17
mul  $18, $18, $19
end_if:
```

- MIPS hardware makes register $0
  (also called $zero) contain 0
  ALWAYS.
- Version I is
  
  \[
  \frac{4 \text{ instructions}}{3 \text{ instructions}} - 1 = 33% \tag{1}
  \]

  LARGER (code size) than Version II.
  (One more jump; jumps are SLOW)
# Let’s add some numbers

```
.text # Let’s write INSTRUCTIONS
.globl __start # system should know __start
__start:      # Execution starts here..
    li  $s0, 45  # reg. $s0=$16 contains 45
    add $s0, $s0, 32  # add in 32 to $s0
    add $s0, $s0, -3  # subtract 3 from $s0
    add $a0, $0, $s0  # 3 register add
        # our sum is now in $a0
    li  $v0, 1
        # $v0 now has 1, the code for print_int
syscall     # prints the int in $a0
```

See Waldron’s Table 3.1 or PH Figure A.10 for MIPS system register use CONVENTIONS.

See Waldron’s Table 3.2 or PH Figure A.17 for SPIM/XSPIM toy SYSTEM CALLS (actions syscall causes).
## hello2.a  hello world; demo of lw instruction

.........
la $a0,str  # put string address into a0
li $v0,4  # system call to print
syscall  # out a string
lw $s0, 0($a0)  #load from addr of "hell"
lw $s1, 4($a0)  #load from addr of "o wo"
addi $a0, $a0, 4  #compute addr of "o wo"
lw $s2, 0($a0)  #load from addr of "o wo"
syscall  #What does this print??

.data
str: .asciiz "hello world\n"

## hello.a  - prints out "hello world"
##
## a0  - points to the string
##
#******************************************************************************
#
#  text segment
#
#******************************************************************************

.text
.globl __start
__start:  # execution starts here
la $a0,str  # put string address into a0
li $v0,4  # system call to print
syscall  # out a string
li $v0,10
Slide 24

syscall  # au revoir...

Slide 25

;.............................................
#  #
#    data segment  #
#  #
;.............................................

.data
str: .asciiz "hello world\n"
##
## end of file hello.a
```assembly
.temp.a ask user for temperature in Celsius,
## convert to Fahrenheit, print the result.
##
## v0 - reads in celsius
## t0 - holds Fahrenheit result
## a0 - points to output strings
##
Slide 26

#
#
# text segment
#
#
Slide 26

.text
.globl __start
__start:
la $a0,prompt    # print prompt on terminal
li $v0,4

.syscall

Slide 27
li $v0,5         # syscall 5 reads an integer
.syscall
```
mul $t0,$v0,9  # to convert, multiply by 9,
div $t0,$t0,5  # divide by 5, then
add $t0,$t0,32  # add 32

la $a0,ans1    # print string before result
li $v0,4    
syscall

move $a0,$t0    # print result
li $v0,1
syscall

la $a0,endl    # system call to print
li $v0,4      # out a newline
syscall

li $v0,10
syscall      # au revoir...

Slide 29

.data
prompt:    .asciiz "Enter temperature (Celsius): "
ans1:       .asciiz "The temperature in Fahrenheit is "
endl:       .asciiz "\n"

##
## end of file temp.a