Assemblers

Kevin Walsh
CS 3410, Spring 2010
Computer Science
Cornell University

See: P&H Appendix B.1-2
T: ADDI r4, r0, -1
   BEQ r3, r0, B
   ADDI r4, r4, 1
   LW r3, 0(r3)
   J T
   NOP
B: ...

L: LW r5, 0(r31)
   ADDI r5, r5, 1
   SW r5, 0(r31)
   ...

JAL L
   nop
   nop
```c
int x = 10;
x = 2 * x + 15;
```

```assembly
addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15
```

Binary code:
```
00100000000000101010000000000001010
00000000000000101010010100001000000
001000001010010100000000000001111
```
```
T: ADDI r4, r0, -1
BEQ r3, r0, B
ADDI r4, r4, 1
LW r3, 0(r3)
J T
NOP
B: ...
```

```
  ...  
  001000
  000100
  001000
  100011
  000010
  00000000000000000000000000000000
  ...  
```
Q: How to resolve labels into offsets and addresses?

A: Two-pass assembly

- 1st pass: lay out instructions and data, and build a *symbol table* (mapping labels to addresses) as you go
- 2nd pass: encode instructions and data in binary, using symbol table to resolve references
\[
\begin{array}{l}
\ldots \\
JAL \ L \\
nop \\
nop \\
L: LW r5, 0(r31) \\
ADDI r5, r5, 1 \\
SW r5, 0(r31) \\
\ldots \\
\end{array}
\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>001000000010000000000000000000100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00000000000000000000000000000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00000000000000000000000000000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000111111110011010000000000000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>000010001010010100000000000000001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00000000000000000000000000000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example 2
.text 0x00400000 # code segment

...  
ORI r4, r0, counter  
LW r5, 0(r4)  
ADDI r5, r5, 1  
SW r5, 0(r4)  
...

.data 0x10000000 # data segment

counter:
  .word 0
Lessons:

• Mixed data and instructions (von Neumann)
• ... but best kept in separate segments
• Specify layout and data using assembler directives
• Use pseudo-instructions
Pseudo-Instructions

NOP # do nothing

MOVE reg, reg # copy between regs

LI reg, imm # load immediate (up to 32 bits)

LA reg, label # load address (32 bits)

B label # unconditional branch

BLT reg, reg, label # branch less than
Assembler:

- assembly instructions
- pseudo-instructions
- data and layout directives
- executable program

Slightly higher level than plain assembly

e.g: takes care of delay slots

  (will reorder instructions or insert nops)
Q: Will I program in assembly?
A: I do...

• For kernel hacking, device drivers, GPU, etc.
• For performance (but compilers are getting better)
• For highly time critical sections
• For hardware without high level languages
• For new & advanced instructions: rdtsc, debug registers, performance counters, synchronization, ...