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: ...

... JAL L
    nop
    nop
L:  LW r5, 0(r31)
    ADDI r5, r5, 1
    SW r5, 0(r31)
...
```c
int x = 10;
x = 2 * x + 15;
```

```
addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15
```
T: ADDI r4, r0, -1
    BEQ r3, r0, B
ADDI r4, r4, 1
LW r3, 0(r3)
J T
NOP
B: ....

...
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
JAL L
nop
\text{\textcolor{red}{nop\ \texttt{Counter} = 0}}
L: LW r5, 0(r31)
ADDI r5, r5, 1
SW r5, 0(r31)
\ldots

\begin{array}{|c|}
\hline
\ldots \\
00100000001000000000000000100 \\
000000000000000000000000000 \\
000000000000000000000000000 \\
1000111111100101000000000000000 \\
0001000101001010000000000000001 \\
000000000000000000000000000000000 \\
\hline
\end{array}
.text 0x00400000 # code segment

...  LA

ORI r4, r0, counter

LW r5, 0(r4)

ADDI r5, r5, 1

SW r5, 0(r4)

...

.data 0x100000000 # 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

\[
\begin{align*}
& \text{SLT} \\
& \text{B not} \\
\end{align*}
\]
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, ...

\[ x86 - \text{Read Time Stamp Counter} \]