Computer Science 210

Computer Systems 1

2007 Semester 1

Lecture Notes Part 2

The Assembly Process

Lecture 11
27 Apr 07

James Goodman

Department of Computer Science

Recommended Readings

For today's lecture

- Hutton's Notes, Chapter 12: assembling and disassembling

In-class Test

- Test Thursday 3May
- During class (this room)
- Coverage: lectures and assignments up to term break

Translation vs. Interpretation

- A program written in language L defines a "machine"
- Problem:
  - We have a program written in language L1
  - We have a computer that understands how to execute language L2
  - How to "execute" program?
- Solution 1: Compilation/Translation/Assembly
  - A "compiler" takes as input a program written in L1 and creates a program written in L2
- Solution 2: Interpretation/Simulation/Emulation
  - A "program" takes as input a program written in L1 and walks through its execution, taking input and creating output as if it were an L1 computer.

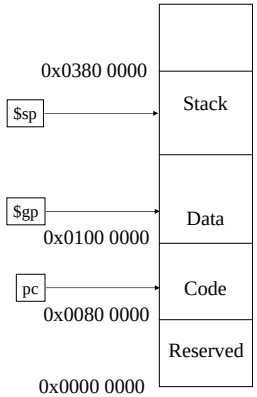
The assembly process: Overview

- A computer understands machine code
- People (and compilers) write assembly language
  - Today it's usually compilers
- Goal: create a file describing exactly what memory should look like before starting execution of a program
- Assembly: the process of translating a program written in assembly language into a program written in machine language.
  - Machine language is specific to a computer
  - Need to create memory image that includes
    - instructions (including links to libraries, kernel, etc.)
    - data (constants, static variables, dynamic variables

Steps in Assembly

- Pass 1: Scan program and parse
  - Identify declarations of instructions and static data
  - identify pseudo-instructions
  - Allocate space for instructions and data
  - Recognize labels
    - Define address if possible
    - Remember for future reference: Symbol Table
  - For data
    - Allocate space
    - Associate label with address
    - Generate appropriate representation
- Pass 2: Rescan and generate code
  - Translate instruction to machine code

A Picture of Memory



Program to Assemble

```
data {
    number:
        quad      32769;
    string1:
        asciiz: "Hello!\n"
    } data
code {
public enter:
        beq       $t0, yyy;
    xxx:
        ldq       $s0, ($t0);
        subq      $s0, 1
        bne       $s0, xxx;
    yyy:
        addq      $zero, 123, $t0
        br        xxx;
        clr       $a1;
        ldiq      $a0, 0;           / CALLSYS_EXIT;
        call_pal 0x83;              / CALL_PAL_CALLSYS;
    } code
```


ASCII Characters

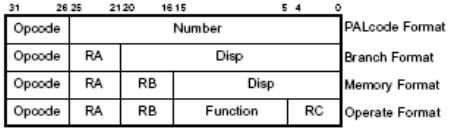
H	0x48
e	0x65
l	0x6c
l	0x6c
o	0x6f
!	0x21
\n	0x0a
0	ox00

Symbol Table

Symbol	Address
number	0x0100 0000
string1	0x0100 0004
xxx	0x0080 0004
yyy	0x0080 0010

Instruction Formats

Figure 1–1: Instruction Format Overview



– From The Alpha Architecture Handbook, Compaq Computer Corporation, 1998.

Opcode Assignment

Instruction	Format	Opcode	Function code
beq	Branch	0x39	
bne	Branch	0x3d	
br	Branch	0x30	
bis	Operate	0x11	0x20
ldq	Memory	0x29	
addq	Operate	0x10	0x20
subq	Operate	0x10	0x29
cal_pal	PALcode	0x00	

Register Names

$0	$v0
$1-$8,	$t0-$t9
$9-$14	$s0-$s5
$15	$fp
$16-$21	$a0-$a5
$22-$25	$t8-$t11
$26	$ra
$27	$pv
$28	$at
$29	$gp
$30	$sp
$31	$zero (special)

Program to Assemble

```
data {
    number:
        quad      32769;
    string1:
        asciiz: "Hello!\n"
    } data
code {
public enter:
        beq       $t0, yyy;
    xxx:
        ldq       $s0, ($t0);
        subq      $s0, 1
        bne       $s0, xxx;
    yyy:
        addq      $zero, 123, $t0
        br        xxx;
        clr       $a1;
        ldiq      $a0, 0;           / CALLSYS_EXIT;
        call_pal 0x83;              / CALL_PAL_CALLSYS;
    } code
```


Working Assembly

```
                    0x32    $t0=1                +3
0x0080 0000   1100 10|00 001|0 0000 0000 0000 0000 0011 0000
                    0x29    $s0=9   $t0=1       +0000
0x0080 0004   1010 01|01 001|0 0001|0000 0000 0000 0000 0000
0x0080 0008
  ...

0x0100 0000   0000|0000|0000|0000|1000|0000|0000|0001
0x0100 0004   0000 0000 0000 0000 0000 0000 0000 0000
0x0100 0008   0110 1100|0110 1100|0110 0101|0100 1000
0x0100 000c   0000 0000|0000 1010|0010 0001|0110 1111
0x0100 0010
  ...
```


Assembled Code

```
00800000  e4200003      beq       $t0, .main.yyy
00800004  a5210000      ldq       $s0, +0000($t0)
00800008  41203529      subq      $s0, 01, $s0
0080000c  f53ffffd      bne       $s0, .main.xxx
00800010  43ef7401      addq      $zero, 7b, $t0
00800014  c3fffffb      br        $zero, main.xxx
00800018  47ff0411      bis       $zero, $zero, $a1
0080001c  a61d0000      ldq       $a0, +0000($gp)
00800020  00000083      call_pal  0000083
00800024  00000000      call_pal  0000000

                     .main.number:
01000000  00008001        Hex 8001
01000004  00000000
                     .main.string1:
01000008  6c6c6548 Hell   Hex a216f6c6c6548
0100000c  000a216f o!??
```