Announcements

- Homework #7 is due on Thursday
  - clarification on problem 3a (end of fetch cycle is T_2)
- Midterms are graded and were returned
  - exams contain summary of grades to date
- Programming assignment, part “A” was handed out
- Reading for this lecture Chapter 6 (up to 6-7)
  - can skip section 6-4
- For next time, print out the SPIM manual.

Assembly Language

- Provides a (slightly) higher level of abstraction
  - Programs are still written one instruction at a time.
  - provides a more symbolic way to write a program.
- Assembly Language Statement has:
  - label - symbolic names
    - for data and instructions
  - instruction field
    - Opcode (e.g.... ADD)
    - operand
      - can be a label
      - can be a number
    - indirect flag (I)
  - comment fields
    - a “/”

Pseudo Instructions

- look like instructions, but do not result in executable code.
- can have operands too
- come in four types:
  - ORG N — next statement goes into memory loc N
  - END — end of assembly language program
  - DEC N — fill memory with N (a signed decimal #)
  - HEX N — fill memory with N (a hex number)

Sample Assembly Program

```
        ORG 100        / program starts at 100
        LDA SUB        / load # to subtract
        CMA            / complement AC
        INC            / inc ac - now have -SUB
        ADD MIN        / add min AC = MIN-SUN
        STA DIF        / store result to DIF
        HLT            / we are done, stop before
MIN,    DEC 83         /   we hit the data
SUB,    DEC -23
DIF,    HEX 0          / result goes here
        END            / end of the program
```

Converting Assembly to Machine Code

- Need to read each statement
  - convert instruction to correct form
    - lookup the address of the operand
    - set the indirect flag
  - convert pseudo-instructions
    - org — change next address
    - end — stop
    - dec, hex — fill memory
- Need to Maintain Symbol Table
  - maps labels to addresses
  - what happens if we lookup a symbol and it is not found?
    - could be an error
    - could be a symbol we haven’t gotten to yet

Converting C to Assembly Language

- Requires a compiler
- Programs tend to get much longer during conversion
  - one C statement becomes many assembly statements
  - called a one to many translation (assembler to machine code is really one to one)
- Consider a simple C program:

```
main() {
    int sum, I, A[100];
    sum = 0;
    for (I=0; I < 99; I++) {
        sum += A[I];
    }
}
```

Assembly Language Version

```
          ORG 100
          CLA            // clear AC for sum
LOOP,     ADD ADS I      // use ADS to point to array
          ISZ ADS        // add one to ADS for next element
          ISZ I          // add one to our counter, skip next if done
          BUN LOOP       // branch back to the top of the loop
          STA SUM        // store the sum
          HLT
ADS,      HEX 150        // 150 is the base address of the array
I,        DEC -100       // counter goes from -100 to 0
          END
```

Writing a Subroutine

- Write a subroutine (SH4) that:
  - shifts the value in the AC 4 bit positions to the left
  - the four low bits of the AC should be zero
- Write a program that:
  - take the contents of a memory location X and shifts it by 8 bits to the left
  - uses two calls to our subroutine SH4 to do this

Subroutine Example

- Calling side
  - Set up arguments
  - use BSA to call routine
  - save return value

```
        ORG 100
        LDA X
        BSA SH4
        BSA SH4
        STA X
        HLT
X, HEX 4321
```

- Callee Side
  - save room for return addr
  - perform operation
  - return via the return address

```
SH4,    HEX 0
        CIL
        CIL
        CIL
        CIL
        AND MASK
        BUN SH4 I
MASK,   HEX fff0
        END
```

Midterm #2 Results

- Average Score: 57
- Standard Deviation: 20

Question	Average
1	12
2	8.3
3	15
4	8.9
5	8.5
6	5.2

24 hour waiting period on all appeals