Assembly: Review

Lecture 14
Definition

- Recall: Translation
  - A source program is translated into a target program
  - Each language corresponds to an abstract machine
  - Source program is not directly executed

- When source is a symbolic representation of machine language:
  - Source language is ___________
  - Translator is ___________

- When source is higher-level, translator usually called a ___________
Advantage of Assembly

- Over machine code
  - Easier to remember/read mnemonic operations vs actual opcodes
  - Easier to use symbolic addresses

- Over higher-level languages
  - Access to full capabilities of the machine at the lowest level
  - Speed? Not generally true because:
    - Optimizing compilers
    - Algorithm design and insights are more significant
Best of Both Worlds

- Systems programming often done in a language like C
- Syntax of a higher-level language
- Control of lower-level concerns, like memory allocation
So Why Bother With Assembly?

- Might need to write/read/maintain some critical bit of assembly code
- Similar to (but simpler than) compiler
- Good way to learn about computer architecture
- Insights into workings of higher-level languages
- The world still needs compilers, and therefore assembly too
Assembly: Syntax

- One statement / line
- Four fields (not all are necessary):
  - Label
  - Operation
  - Operands
  - Comment
- Example
  Test  BRZ Loop  ;if R1=0 repeat
Label Field

- Symbolic name for the instruction address
- Clarifies branching to a particular instruction
  - BRNP Loop1
- Clarifies target for loading/storing data to memory
  - LD R3, Sum
- Often severely limited in length
Operation Field

- Mnemonic for a machine instruction
  - Eg, ADD, SUB, BRZ

- Mnemonic for a “pseudo operation”
  - Eg, .FILL
  - More about these in a moment
Operand Field

- Arguments to the function
- Registers, immediate data, address used by the instruction
  - What to add, where to branch, where to store, etc.
- Information used by pseudo operations
  - Information for the assembler to produce the object file
  - Program name, how much space to save, etc.
Comment Field

- No effect on assembler
  - No difference in resulting object file
  - No semantic impact on program

- But huge impact on legibility
  - Strictly for human consumption
Pseudo Operations

- Unlike instructions, pseudo ops do not have a corresponding machine instruction in the ISA
- Give information to assembler itself
  - “assembler directive”
  - Control various aspects of resulting object file
- Uses
  1. Segment definition
  2. Symbol definition
  3. Memory initialization
  4. Storage allocation
Segment Definition

- Recall object file header & end records
  - Segment name
  - Segment load address
  - Segment length
  - Initial execution address

- All of this information comes directly from pseudo ops
  - Exception: __________
Segment Definition

- Two important pseudo ops:
  - .ORIG (for “origin”)
  - .END

- Example
  ```
  MainP .ORIG x300A
  LD R0,x126
  ...
  .END x300E
  ```

- What are the header & end records of the resulting object file?
x300A | LD R0, x136
---|---
x300B | 
---|---
x300E | 
---|---

Header record: H
End record: E
Header record: HMainP_300A????
End record: E300E
Symbol Definition

- A label creates a symbol
- Symbol is implicitly defined to be the address of the instruction
- Example: What is the value of symbol Test?

```
Hello  .ORIG  x300A
   LD     R0,x126
Test   BRZ    x147
```
Explicit Symbol Definition

- Symbols can also be defined explicitly
- Pseudo Op:
  - .EQU (“equate”)
- Example
  - `STEP .EQU #2 ;Skip odd indices`
- Symbols are then used as program constants
  - `ADD R1,R1,STEP`
  - `HALT .EQU x25`
  - `TRAP HALT`
Summary

- Symbolic machine code
- Advantages/disadvantages
- Basic syntax
- Pseudo Operations
  - Segment definition
  - Symbol Definition