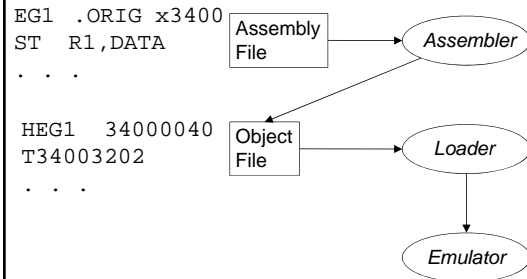
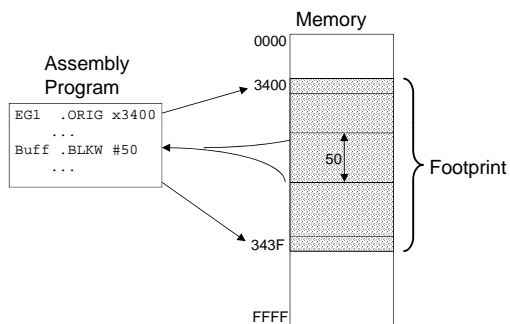


Lecture #15

“Big Picture”: Labs 1-3



“Big Picture”: Lab 2



Lab 2

- Assembler does *not* need to keep this (potentially huge) array/footprint
- Instead, use tables (symbol, literal, ...) and location counter
- Generate *object file* only (much smaller)

Example

```

1  Prog  .ORIG x3020
2  HALT  .EQU  x25
3  Begin LD   R0,N      ;R0 <- #13
4         LD   R1,#16   ;R1 <- #16
5         ST   R0,Ans   ;M[Ans]<-R0
6         TRAP HALT
7  N     .FILL #13
8  Ans   .BLKW 1
9         .END  Begin

```

Example: Object File

```

HProg 30200007
T30202024
T30212226
T30223025
T3023F025
T3024000D
T30260010
E3020

```

Lecture #16

Assembler Tasks

1. Parse assembly instructions
 - check for syntax
 - tokenize string
2. Assign addresses to instructions
 - maintain “location counter” (LC)
 - LC = eventual location in memory of this instruction

Assembler Tasks II

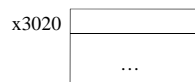
3. Generate machine code
 - evaluate mnemonic instruction
 - replace with opcode
 - evaluate operand subfields
 - replace symbols & literals with value
 - concatenate to form instruction
4. Process pseudo-ops
 - generate header record
 - evaluate .FILL, .STRZ, ... etc

Assembler Tasks III

5. Write output file
 - header & text records
- *Nothing here seems all that hard...*
 - So let's walk through the assembly of our simple example program

First Attempt

- Read each input line and generate machine code
- Line 1
 - information for header
 - but not enough for full header record
 - we do know:



First Attempt II

- Line 2:
 - information for assembler
 - symbol “HALT” set to x25
- Line 3:
 - yeah! An instruction to translate! (LD R0,N)
 - yields:



First Attempt - Difficulties

- Problem:
- Lines 4 & 5:
 - same problem
 - do not yet know address for =#16, or Ans
- Solution:
 -
 -

Now let's see some basic data structures for assemblers...

Machine Op Table

Mnemonic Name	Opcode	Instruction Size	Instruction Format

- Static (doesn't change during computation)
- For the (simple) abstract machine:
 - all opcodes are 4 bits
 - all instructions are same size (i.e., 1 word)
 - formats do differ (e.g., Op R,R,imm)

Machine Op Table II

- But this need not be the case in general
 - expanding opcodes
 - common instructions have short opcodes (eg, 0110)
 - less common ones are longer (eg, 1110110)
 - variable instruction length
 - e.g., “branch relative”, where $PC \leftarrow PC + \text{operand}$
 - could be near (same page), operand is 9 bits
 - could be far, operand is 25 bits (instruction uses 2 words)



Pseudo Op Table

Mnemonic	Length	Format

- Also a static table
- Some “lengths” are 0, others are 1, or variable

Location Counter

- Eventual address of this instruction
- Initialized with _____
- Increment with each instruction
 - see _____
- Increment with (some) pseudo-ops
 - see _____

Symbol Table

Name	Value	Other Stuff

- Pass #1
 - each symbol is *identified*
 - every time a new symbol is seen (ie a label), insert it into the Symbol Table
 - if it's already there?

Symbol Table II

- Pass #1 (continued)
 - each symbol given a *value*
 - explicit assignment (e.g. HALT .EQU x25)
 - easy!
 - just put the value of operand into the table
 - implicit assignment (e.g. X .FILL #13)
 - must know the address of this instruction
 - so, keep track of addresses as program is scanned
 - use "location counter" (LC)

Symbol Table III

- Pass #2
 - symbols in operands replaced with their value
 - look up symbol in Symbol Table
 - if there, replace with value
 - note: value will be larger than field in instruction!
 - address is 16 bits, but poffset is only 9
 - solution:
 - if not there,
 - (Q: how could the symbol possibly not be there?)

Literal Table

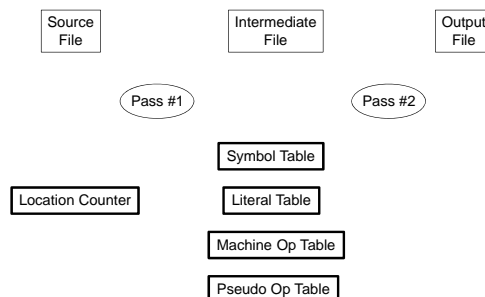
Name	Address	Value	Size	Other Stuff

- Pass #1
 - literals are identified and placed in the table
 - "name", "value", and "size" fields updated
 - duplicates can be eliminated

Literal Table II

- To complete pass #1:
 - literals are added to the end of the program
 - "address" field can now be calculated
- Pass #2:
 - literals in instructions are replaced with the _____ field from the Literal Table
 - what if the literal is not in the table?

Information Flow (Passes 1/2)



Lecture #17

To Ponder

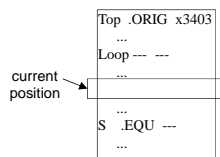
- CNN reported recently that about 13,000 Visa customers found on their last statements a charge for \$23,148,855,308,184,500
- What went wrong?
- (These customers were also charged a \$15 “overdraft” fee!)

- $23,148,855,308,184,500 \times 100$
 $= 2,314,885,530,818,450,000$
 $= \text{x2020202020201250}$
- And x20 is ASCII blank!
- (x1250 is ?? maybe \$46.99?)
- Aside: apparently Visa uses 64 bit integers

Two Pass Assembler: Limitations

- Q: Does our 2-pass approach solve all forward-reference problems?
- A: no! Something is still broken...
- Hint:
 - what is the *key invariant* (w.r.t. symbols) during pass #1?

Pass #1 Invariant



- Key invariants:
- So what could go wrong?
 - ie how might maintaining this invariant be difficult?
- How could this happen?

Forward Reference Restriction

- Consider

Y	.EQU	0
X	.EQU	Y

X	.EQU	Y
Y	.EQU	0
- To avoid this trouble, impose a restriction:
 -
- What about .BLKW? Is a forward symbol allowed as the operand?
 -

Example

```
1  Prog  .ORIG x3020
2  HALT  .EQU  x25
3  Begin LD   R0,N      ;R0 <- #13
4         LD   R1,#16   ;R1 <- #16
5         ST   R0,Ans   ;M[Ans]<-R0
6         TRAP HALT
7  N     .FILL #13
8  Ans   .BLKW 1
9         .END  Begin
```

502

Lecture #18

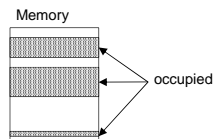
Relocation

Absolute Programs

- Programmer decides *a priori* where program will reside
 - e.g., Prog .ORIG x3176
- But memory is *shared*
 - concurrent users, concurrent jobs/processes
 - several programs loaded simultaneously

Absolute Programs: Limitation

- At any given instant:



- Picture is *dynamic*
 - jobs are scheduled
 - jobs complete

} We cannot predict what this picture will look like!!

Absolute Programs: Limitation II

- Would like the loading to be *flexible*
 - decide at *load* time where it goes! (not at _____ time)
 - this decision is made by _____
- What the programmer wants:
“find a free slot in memory that is big enough to fit this program”

Motivating Relocation: Example

```

Prog .ORIG x0
X .FILL Y
Halt .EQU x25
Start LD R1,X
      ST R1,Y
      TRAP Halt
Y .BLKW 1
  .END Start
    
```

- In memory, this program appears as:

x0000	0	0	0	4
x0001	2	2	0	0
x0002	3	2	0	4
x0003	F	0	2	5
x0004	0	0	0	0

One Slight Change

```

Prog .ORIG x0
X .FILL Y
Halt .EQU x25
Start LD R1,X
      ST R1,Y
      TRAP Halt
Y .BLKW 1
  .END Start
    
```

```

Prog .ORIG x4058
X .FILL Y
Halt .EQU x25
Start LD R1,X
      ST R1,Y
      TRAP Halt
Y .BLKW 1
  .END Start
    
```

One Slight Change

- Appearance of new program in memory:

x4058	4	0	5	C
x4059	2	2	5	8
x405A	3	2	5	C
x405B	F	0	2	5
x405C	0	0	0	0

```

Prog .ORIG x4058
X .FILL Y
Halt .EQU x25
Start LD R1,X
      ST R1,Y
      TRAP Halt
Y .BLKW 1
  .END Start
    
```

Compare the Old and the New

Load Address = _____

Load Address = _____

x4058	4	0	5	C
x4059	2	2	5	8
x405A	3	2	5	C
x405B	F	0	2	5
x405C	0	0	0	0

x0000	0	0	0	4
x0001	2	2	0	0
x0002	3	2	0	4
x0003	F	0	2	5
x0004	0	0	0	0

Another Slight Change

```

Prog .ORIG x71A5
X .FILL Y
Halt .EQU x25
Start LD R1,X
      ST R1,Y
      TRAP Halt
Y .BLKW 1
  .END Start
    
```

- In memory, this program appears as:

x71A5	7	1	A	9
x71A6	2		A	5
x71A7	3		A	9
x71A8	F	0	2	5
x71A9	0	0	0	0

Relocation

- The loader must update *some* (parts of) text records, but *not all*
 - after load address has been determined
- The assembler does 2 things:
 - assemble with a load address of 0
 - tell the loader which parts will need to be updated

Convention

- To denote a relocatable program, omit the operand of .ORIG
Prog1 .ORIG x4096 (absolute)
Prog2 .ORIG (relocatable)

Modification Records

- One approach: define a new record type

Tag	Location
-----	----------

- For our example:

```
HProg 00000005
T0000 0004
T0001 2200
T0002 3204
T0003 F025
E0001
```

Modification Records II

- We could add the following records to the object file:
M0000
M0001
M0002
- Also need to indicate size of quantity being relocated:
 - Two sizes: 9 bit poffset and 16 bit full word
M0000_16
M0001_9
M0002_9

Modification Records III

- One disadvantage of this approach:
 -

Alternative: Bit Masks

- Use 1 bit / memory cell (or byte)
 - bit value is 0 means no relocation necessary
 - bit value is 1 means relocation necessary
- Size of relocation data independent of number of records needing modification
 - but more densely packed (1 bit / text record)
- Hard to read (debug, grade,...)

Compromise (For Our Machine)

- Change the syntax of a text record
- Flag modification with an "M" at the end of the record
 - Distinguish long and short modifications (M1, M0)
- Example
H-----
T0000---- M1
T0001---- M0
T0002---- M0
T0003----

Beware: Page Boundaries

```

Prog .ORIG x65FF
X .FILL Y
Halt .EQU x25
Start LD R1,X
      ST R1,Y
      TRAP Halt
Y .BLKW 1
  .END Start
    
```

- In memory, this program appears as:

x65FF	6	6	0	3
x6600	2	3	F	F
x6601	3	2	0	3
x6602	F	0	2	5
x6603	0	0	0	0

Recall

```

Prog .ORIG x71A5
X .FILL Y
Halt .EQU x25
Start LD R1,X
      ST R1,Y
      TRAP Halt
Y .BLKW 1
  .END Start
    
```

- In memory, this program appears as:

x71A5	7	1	A	9
x71A6	2	3	A	5
x71A7	3	2	A	9
x71A8	F	0	2	5
x71A9	0	0	0	0

Kinds of Data

- Our machine has two flavors of data:
 - relative (to the load address)
 - absolute
- The first must be modified, the second not
- Let's look at how these kinds arise...

Symbols

- Some are relative:
 - e.g.,
- Some are absolute:
 - e.g.,
- Symbol Table

Name	Value	Relative?

Symbols: Rules

- A symbol is absolute if and only if it is defined by:
 - _____ , or
 - _____
- (we'll see another way later)

Literals

- Our machine does not have relative literals
- Other machines allow a special literal, =*, to mean "current location counter"
 - e.g., LD R1,=*
 - such a literal is *relative*, others are *absolute*

Name	Value	Address	Relative?
Star1			
=6			

Literals

- With literals, “relative” refers to whether or not the *value* is relative
 - the *address* of a literal is always relative!

Symbol Table with Relocation Information: Example

```
X .FILL Y
Halt .EQU x25
Start LD R1,X
      ST R1,Y
      TRAP x25
Y .BLKW 1
  .END Start
```

- After pass #1, the symbol table is:

Name	Value	Relative?