Lecture 7
Separate compilation

Computing platforms

Novosibirsk State University
University of Hertfordshire

D. Irtegov, A.Shafarenko

2018

The problem

- In previous lecture we learned how to create subroutines.
- There are many kinds of subroutines good for reuse, like multiplication, division, string operations, etc
- How to actually do the reusing?

Solutions: #include statement

- Not present in CdM-8 assembler
- Slow on big programs
    - Not an issue for CdM-8
    - But bad for real computers
- label name conflicts (“name space pollution”)
- What happens if several modules have conlicting asect directives?

Separate compilation and linking

- Historically, was invented independently and slightly before of assembler
- Now, assemblers and linkers are considered a tightly-coupled elements of toolchain
- By default, assembler produces not a final memory image, but some intermediate format, known as *object file*
- Linker collects several object files and links them into final memory image (executable file)

History of linkers and library routines

- Code reuse was introduced by Grace Hopper in 1944 when programming a Harvard Mark I computer (aka IBM ASCC)
- Mark I was a sequential (not von Neumann) computer
- Sequential computer program contains no addresses
- Only way to implement a loop is to unroll it
  (like we did with multiplication routine in prev. lecture)
- No conditional statements nor while loops
- You could insert a subroutine in any point of the program, provided that it matches a calling convention

Subroutines on von Neumann computers

- On von Neumann computer, programs contain addresses
  (in assembler they are label references)
- To relocate program in memory, we must recalculate these addresses
- When programming early von Neumann computers (EDVAC, UNIVAC) people tried to recalculate addresses manually, but this took time and produced many errors
- Then, Grace Hopper come with the idea of linker or link editor
  – a program tool to recalculate addresses in library routines
- It was one of the first programs to aid in writing programs

So, let’s go back to CdM-8

- We must avoid using asect directive. We cannot link modules with asects mapping on the same address
- We must designate some labels as externally visible (similar to extern in C)

rsect directive

```
                     #################### section mul, contain 2 subs
                             rsect mul
                     mul>
                     #       computes product of r0 and r1, result goes in r1

00: c2                       save r2
01: 3a                         clr r2
                               while
02: 00                            tst r0
03: ed 09                      stays gt
05: 16                            add r1, r2
06: 88                            dec r0
07: ee 02                      wend
09: 09                         move r2,r1
0a: c6                       restore
0b: d7                       rts

```

rsect directive

- Creates a named relative (relocatable) section
- All labels in this section belong to it
- Some labels can be declared as externally visible
- In CdM-8 this is done by using ‘>’ character instead of ’:’
  - Other assemblers use wide range of other syntaxes
  - Most typical is a directive ‘global’ which declares a label to be global
- A file can contain several rsects
  - More on this later
- R-sect cannot span several files
  - In other assemblers it can

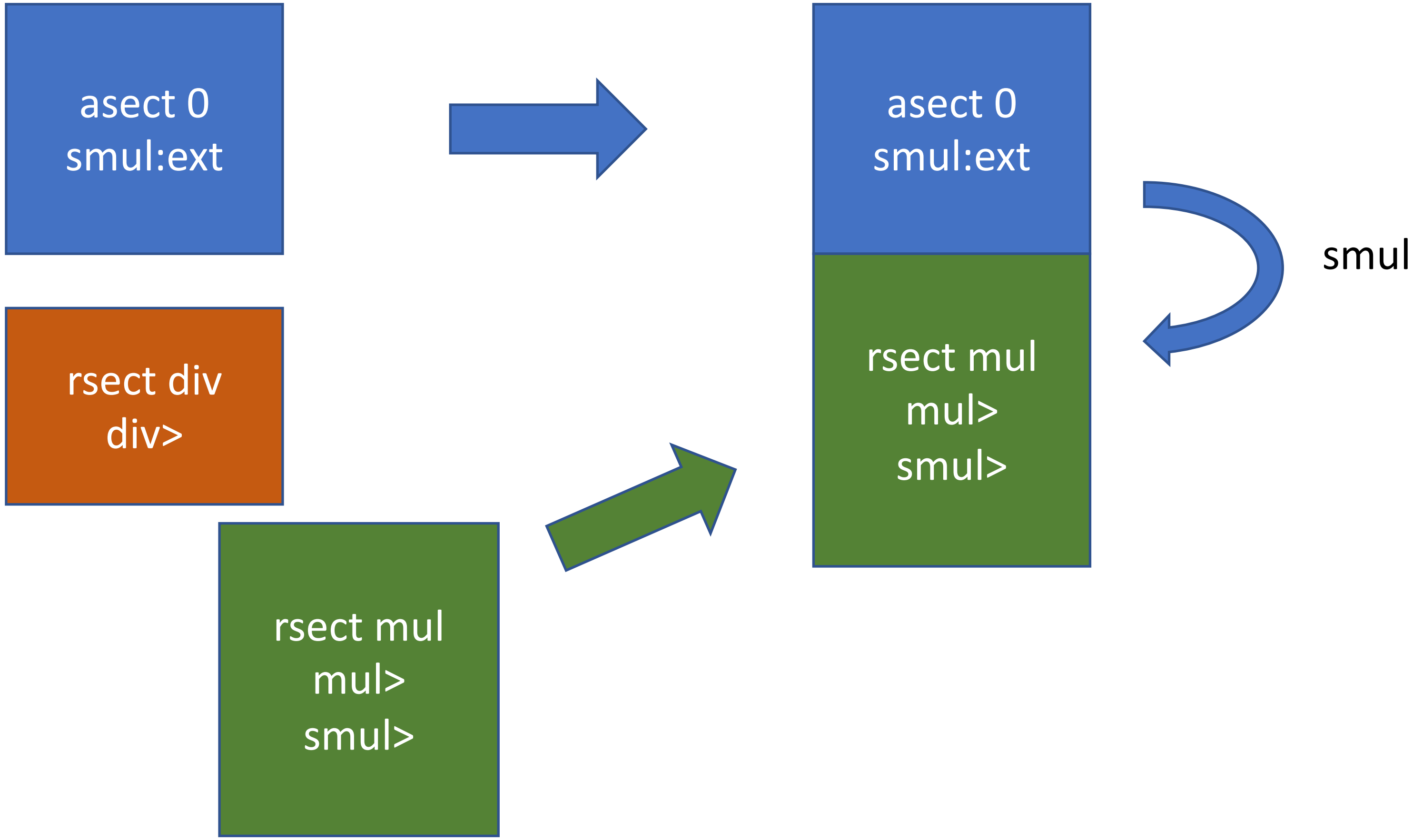
Main program

```
                     # compute -3x+7,
                             asect 0
                     smul:   ext             # declare smul as an external label
                                             # to be defined by an ent elsewhere
00: d0 0b                    ldi     r0,x
02: b0                       ld      r0,r0
03: d1 fd                    ldi     r1,-3
05: d6 00                    jsr     smul
07: d0 07                    ldi     r0,7
09: 11                       add     r0,r1
0a: d4                       halt
0b: 11               x:      dc      17      # example value for testing
                             end
```

What linker does with sections

- First, it allocates a place for asect
- Several asect directives with different start addresses are threated as a single non-contiguous asect
- Second, it finds a places for *referenced* R-sects
- R-sects with no references are excluded from linking
- Third, it relocates R-sects to their places (recalculates addresses)
- Fourth, it writes values of external labels to places where they are referenced (a linking in a strict sence)

A picture



asect 0
smul:ext

rsect div
div>

rsect mul
mul>
smul>

asect 0
smul:ext

rsect mul
mul>
smul>

smul

CdM-8 object file (source and file itself)

```
                  1           asect   0xe0
e0: 03            2  my>      dc 3
                  3  q>
e1: d2 e1         4           ldi r2,q
                  5           rsect   foo
00: 10            6  bar>     add r0,r0
01: d4            7           halt
                  8           rsect main
00: 71            9  main>    cmp r0,r1
01: e8 04        10           bhi z3
03: d5           11           wait
04: d4           12  z3:      halt
                 13           end
```

```
ABS  e0: 03 d2 e1
NTRY q e1
NTRY my e0
NAME main
DATA 71 e8 04 d5 d4
REL  02
NTRY main 00
NAME foo
DATA 10 d4
REL
NTRY bar 00
```

What is REL 02 record?

- It is so called relocation entry.
- Let’s look at this more closely

NAME main

DATA 71 e8 04 d5 d4

REL 02

- Rel 02 points to address field of bhi z3 instruction
- This field must be recalculated when R-sect is relocated

```
e0:  03

e1:  d2  e1

00:  10
01:  d4

00:  71
01:  e8  04
03:  d5
04:  d4
```

```
               asect   0xe0
   my>         dc  3
   q>
               ldi  r2,q
               rsect   foo
   bar>        add  r0,r0
               halt
               rsect  main
   main>       cmp  r0,r1
               bhi  z3
               wait
   z3:         halt
               end
```

Relocation table

- Every R-sect has a relocation table
- In CdM-8 object format it is just list of REL records belonging to a R-sect
- Every REL record is a reference to an address that needs to be relocated (recalculated) according to the actual position of the section
- Some R-sects can have empty relocation table

How it really works

- When assembling a file, assembler creates:
- a symbol table
  - List of all symbols (labels) together with their values
- A cross-reference table
  - List of all places in the code where a specific symbol is referenced
- During a separate compilation, assembler cannot fully build a symbol table
- For external references, it doesn’t know anything about a symbol
- For references to labels in R-sects, you know their offset, but not a final value

Placeholders

- For all references to unresolved symbols, assembler creates
    - A placeholder in the code
    - For relocatable symbols, placeholder contains offset from the R-sect start
    - For external symbols, placeholder can contain anything
    - A reference in cross-reference table (REL for relocatable symbols, XTRN for external)
- When resolving external symbols, linker adds symbol value to the placeholder (this allows references like mul+10)
- When resolving relocatable symbols, linker adds section start to the offset