

ELEC2041

Microprocessors and Interfacing

Lectures 24: Compiler, Assembler, Linker and Loader – I

<http://webct.edtec.unsw.edu.au/>

May 2006

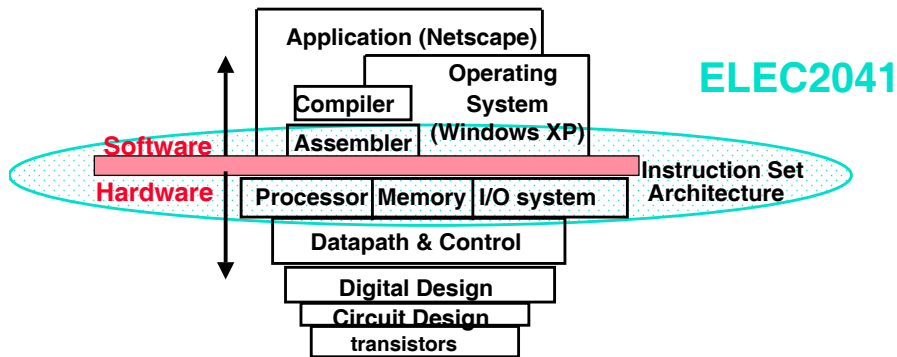
Saeid Nooshabadi

saeid@unsw.edu.au

Overview

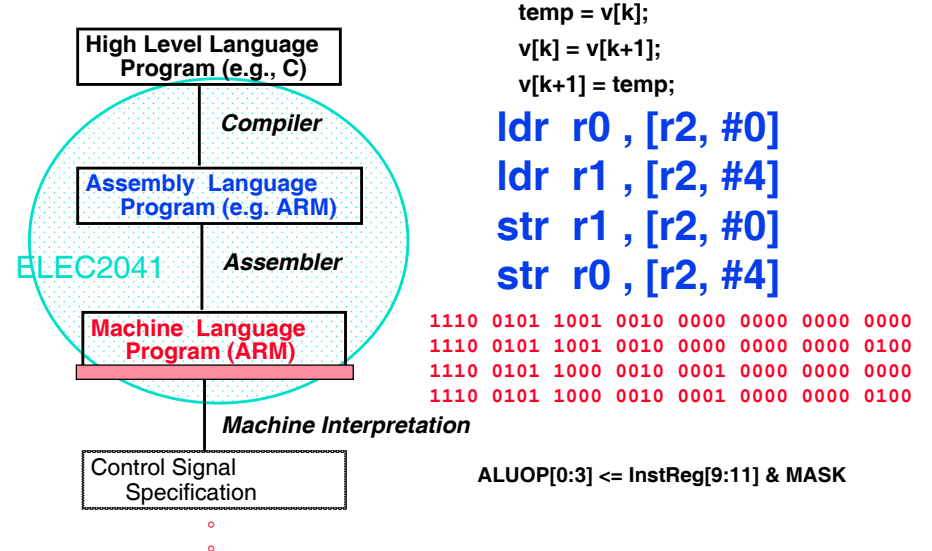
- Compiler
- Assembler
- Linker
- Loader
- Example

Review: What is Subject about?



◦ Coordination of many *levels of abstraction*

Review: Programming Levels of Representation



Review: Stored Program Concept

- **Stored Program Concept:** Both data and actual code (instructions) are stored in the same memory.
- **Type is not associated with data,** bits have no meaning unless given in context

Review: ARM Instruction Set Format

31	2827	1615	87	0	Instruction type						
Cond	0 0 1	Opcode	S	Rn	Rd	Operand2	Data processing / PSR transfer				
Cond	0 0 0 0 0 0 0	A S	Rd	Rn	Rs	1 0 0 1	Rm	Multiply			
Cond	0 0 0 0 1	U A S	RdHi	RdLo	Rs	1 0 0 1	Rm	Long Multiply (v3M / v4 only)			
Cond	0 0 0 1 0	B 0 0	Rn	Rd	0 0 0 0	1 0 0 1	Rm	Swap			
Cond	0 1	I F U B W L	Rn	Rd	Offset			Load/Store Byte/Word			
Cond	1 0 0	F U S W L	Rn	Register List				Load/Store Multiple			
Cond	0 0 0	F U 1 W L	Rn	Rd	Offset1	1	S	H	1	Offset2	Halfword transfer: Immediate offset (v4 only)
Cond	0 0 0	F U 0 W L	Rn	Rd	0 0 0 0	1	S	H	1	Rm	Halfword transfer: Register offset (v4 only)
Cond	1 0 1	Offset									Branch
Cond	0 0 0 1	0 0 1 0	1 1 1 1	1 1 1 1	1 1 1 1	0 0 0 1	Rn	Branch Exchange (v4T only)			
Cond	1 1 0	F U N W L	Rn	CRd	CPNum	Offset					Coprocessor data transfer
Cond	1 1 1 0	Op1	CRn	CRd	CPNum	Op2	0	CRm	Coprocessor data operation		
Cond	1 1 1 0	Op1	L	CRn	Rd	CPNum	Op2	1	CRm	Coprocessor register transfer	
Cond	1 1 1 1	SWI Number									Software interrupt

All Instruction 32 bits

Review: Example Assembly

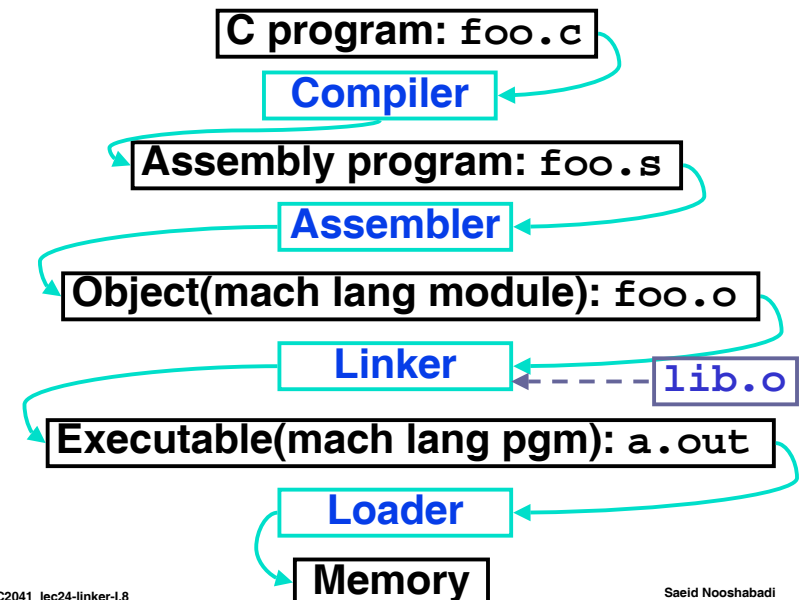
```
sub r2, r3, #1      => 0Xe2432001
sub r2, r3, r4      => 0Xe0432004
b   foo            => 0Xea &foo-----
```

1110	001	0010	0	0011	0010	0000	00000001
1110	000	0010	0	0011	0010	0000	00000100
1110	101	0-----					

14	1	2	0	3	2	0	1
14	0	2	0	3	2	0	0 0 4
14	5	0	?				

$$? = ((pc + 8) - \&foo) >> 2$$

Steps to Starting a Program



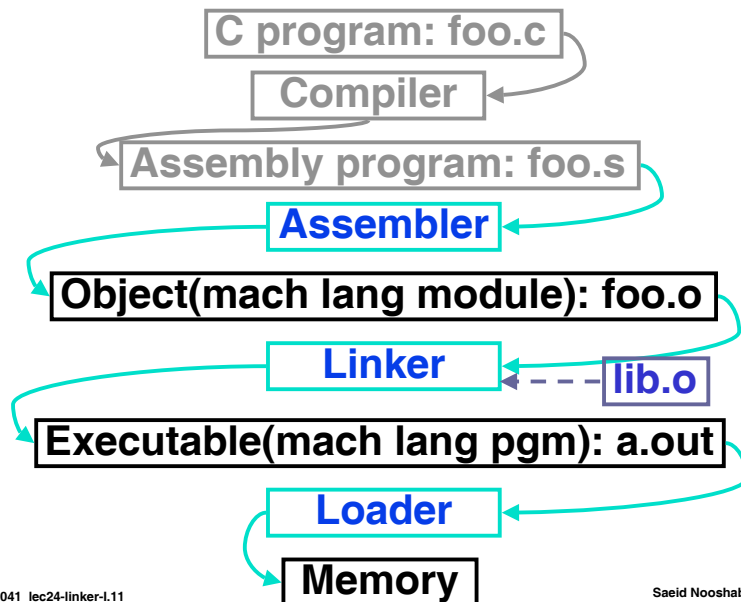
Compiler

- Input: High-Level Language Code (e.g., C, Java)
- Output: Assembly Language Code (e.g., ARM)
- Most Compiler can generate Object code (Machine language) directly

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run

```
extern int posmul(int mlier, int mcand);
int main (void)
{
    char *MESEG = "Multiplication";
    static int a=20,b=18, c;
        c = posmul(a, b);
        return c;
}
```

Where Are We Now?



Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run (#1/3)

```
.data
a.0: .align 2      Data Segment
     .word 20
b.1: .align 2
     .word 18
c.2: .align 2
     .space 4

.section          .rodata
.align 2
.LC0:
.ascii "Multiplication\000"
```

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run (#2/3)

```

.text
.align 2          Text Segment
.global main

main:
stmfd    sp!, {r4,lr}
ldr     r4, .L2      ; MSG
ldr     r3, .L2+4
ldr     r2, .L2+8
ldr     r0, [r3, #0] ; a
ldr     r1, [r2, #0] ; b
bl      posmul
mov     r2, r0
ldr     r3, .L2+12
str     r2, [r3, #0] ; c
mov     r0, r3
ldmfd   sp!, {r4, pc}

```

Addresses of MSG, a, b & c are stored at label .L2

Indirect access to a, b & c

ELEC2041 lec24-linker-1.13

Saeid Nooshabadi

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run (#3/3)

```

.L3:
    .align 2
.L2:
    .word .LC0
    .word a.0
    .word b.1
    .word c.2

```

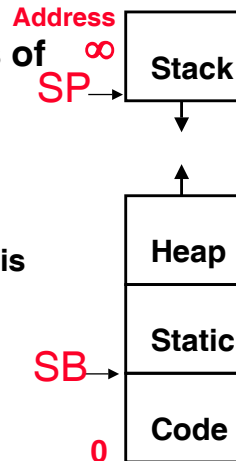
Literal Pool for storing addresses of MSG, a, b & c

ELEC2041 lec24-linker-1.14

Saeid Nooshabadi

What is Assembler?

- Program that translates symbolic machine instructions into binary representation
- encodes code and data as blocks of bits from symbolic instruction, declarations, and directives
- It builds the code words and the static data words
 - loaded into memory when program is run
- what must it do
 - map opcodes, regs, literals into bit fields
 - map labels into addresses



ELEC2041 lec24-linker-1.15

Saeid Nooshabadi

How does Assembler work?

- Reads and Uses **Directives**
- Replace Pseudoinstructions
- Produce Machine Language
- Creates **Object File (*.o files)**

ELEC2041 lec24-linker-1.16

Saeid Nooshabadi

Assembler Directives

- Give directions to assembler, but do not produce machine instructions
 - .text: Subsequent items put in user text segment
 - .data: Subsequent items put in user data segment
 - .globl sym: declares sym global and can be referenced from other files
 - .asciz str: Store the string str in memory and null-terminate it
 - .word w1...wn: Store the n 32-bit quantities in successive memory words

ELEC2041 lec24-linker-1.17

Saeid Nooshabadi

Pseudo-instruction Replacement (#1/6)

- Assembler provide many convenient shorthand special cases of real instructions
 - **nop** → mov, r0, r0
 - **mov r0, #0xffffffff0** → mvn r0, 0xf
 - **ldr/str rdest, label** → load/stores a value at label (address) in the same segment. Converts to
 - **ldr rdest, [pc, #offset]** instruction, where offset is computed by (address@label - [pc + 8]).
 - **Offset** range $\pm 2^{12}$ (± 4 Kbytes)
 - **adr rdest, label** → load address of a label (in the same segment) computed as an offset from PC. Converts to
 - **sub rdest, pc, #imm ; #imm: 8 bit number**
 - **add rdest, pc, #imm ; or rotated version**
 - **#imm** is computed as $-(\text{address@label} - [\text{pc} + 8])$

ELEC2041 lec24-linker-1.18

Saeid Nooshabadi

Pseudo-instruction Replacement (#2/6)

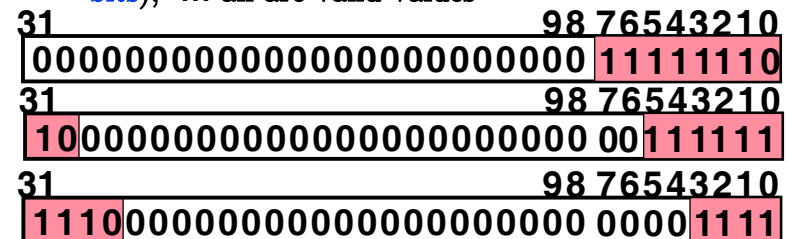
- **ladr rdest, label** → same thing as **adr** when offset value cannot fit in 8 bit rotated format
 - Converts to sequence of two **sub** & **add** instructions
 - If second **sub/add** not needed is replaced by **nop**
- Especially important Pseudo Instructions are for building literals
 - **ldr rdest, =imm32** → load (move) ANY immediate to rdest

ELEC2041 lec24-linker-1.19

Saeid Nooshabadi

Pseudo-instruction Replacement (#3/6)

- Limitation on **mov rdest, #imm** Instruction
 - Any 8-bit value in the range 0 – 255 (0x0 – 0xff)
 - Any 8 bit value in the range 0 – 255 (0x0 – 0xff) rotated to the right two bits at a time.
 - Max rotation = 30 bits
 - Example:
 - 0x000000FE, 0x8000003f (rot. 2 bits), 0xe000000f (rot. 4 bits), ... all are valid values



ELEC2041 lec24-linker-1.20

Saeid Nooshabadi

Pseudo-instruction Replacement (#4/6)

◦ Solution: Use Pseudo Instruction

- Replace `mov rdest, #imm` by `ldr rdest, =imm` => load (move) ANY immediate
- Converts to `mov` or `mvn` instruction, if the constant can be generated by either of these instructions.
- PC relative LDR instruction will be generated to load the `imm` from literal pool inserted at the end of the text segment.

Pseudo-instruction Replacement (#5/6)

◦ Example: Use Pseudo Instruction

```
.text
.align 2
.global main

main:
:
    ldr r2, =4118633130
:

end:
```

◦ Changes to

```
.text
.align 2
.global main

main:
:
    ldr r2, [pc, #offset]
:

end:
.word 4118633130
```

Offset = end - (pc + 8)

Pseudo-instruction Replacement (#6/6)

- **Recall:** `ldr/str rdest, label` → load/stores a value at label (address) in the same segment. Converts to
 - `ldr/str rdest, [pc, #offset]` instruction, where `offset` is computed by $(\text{address@label} - [\text{pc} + 8])$.
 - `offset` range $\pm 2^{12}$ (± 4 Kbytes)
- `ldr rdest1, =label` → load address of ANY label
- PC relative LDR instruction will be generated to load the `address` of the `label` from literal pool inserted at the end of the text segment.
 - `ldr rdest1, [pc, #offset]`
- Next load the `value` at the label by one additional instruction:
 - `ldr/str rdest2, [rdest1]`

Handling Addresses by Assembler (#1/2)

◦ Branches: b, & b1 (branch and link)

b/b1	label
------	-------

- Such branches are normally taken to a label (address) labels at fixed locations, in the same module (file) or other modules (eg. Call to functions in other modules)
- The address of the label is **absolute**

Handling Addresses by Assembler (#2/2)

◦ Loads and stores to variables in static area

- Such addresses are stored in the literal pool by the compiler/Assembler
- The reference to to the literal pool is via PC relative addressing

```
ldr Rdest1, [pc, #offset]
```

```
ldr/str Rdest2, [Rdest1]
```

- Sometimes they are referenced via Static Base Pointer (SB)

```
ldr/str Rdest, [sb, #offset]
```

◦ Loads and stores to local variables

- Such variables are put direct on registers or on stack and are referenced via `sp` or `fp`.

ELEC2041 lec24-linker-1.25

Saeid Nooshabadi

Reading Material

◦ Reading assignment:

- David Patterson and John Hennessy: "Computer Organisation & Design: The HW/SW Interface," 2nd Ed Morgan Kaufmann, 1998, ISBN: 1 - 55860 - 491 - X. ([A.8](#), [8.1-8.4](#))

- Steve Furber: ARM System On-Chip; 2nd Ed, Addison-Wesley, 2000, ISBN: 0-201-67519-6. [chapter 2](#), [section 2.4](#)

ELEC2041 lec24-linker-1.26

Saeid Nooshabadi

Things to Remember

- Compiler converts a single HLL file into a single assembly language file.
- Assembler removes pseudos, converts it can to machine language. This changes each `.s` file into a `.o` file.
- Linker combines several `.o` files and resolves absolute addresses.
- Loader loads executable into memory and begins execution.

ELEC2041 lec24-linker-1.27

Saeid Nooshabadi