CMSC 313
COMPUTER ORGANIZATION
&
ASSEMBLY LANGUAGE PROGRAMMING

LECTURE 12, SPRING 2013

TOPICS TODAY

- Midterm Exam Topics
- Assembling & Linking Assembly Language
- Separate Compilation in C
- Scope and Lifetime

MIDTERM EXAM TOPICS

MIDTERM EXAM

- Thursday, March 14
- In Class
- No Calculators, cell phones, tablets, electronics, ...

MIDTERM FORMAT

- Multiple Choice
- Short responses (e.g., base conversion)
- Trace assembly language program
- Write assembly language program
- Full text of `toupper.asm` available

MIDTERM TOPICS

- **Base Conversion**
- **Data Representation**
  - negative numbers: 2's complement, 1's complement, signed magnitude
  - ASCII
  - little endian vs big endian
- **Intel CPU**
  - Registers
  - Addressing modes
  - Flags
  - Common instructions

Common Instructions

- **Basic Instructions**
  - ADD, SUB, INC, DEC, MOV, NOP
- **Branching Instructions**
  - JMP, CMP, J*cc*
- **More Arithmetic Instructions**
  - NEG, ~~MUL, IMUL,~~ DIV, ~~IDIV~~
- **Logical (bit manipulation) Instructions**
  - AND, OR, NOT, SHL, SHR, SAL, SAR, ROL, ROR, RCL, RCR
- **Subroutine Instructions**
  - PUSH, POP, CALL, RET

MIDTERM TOPICS (CONT'D)

- **Comparison & conditional jump instructions**
  - signed vs unsigned conditional jumps (e.g. ja vs jg)
- **NASM**
  - How to assemble
  - `.data`, `.bss`, `.text` sections
  - `dd`, `dw`, `db`, `resd`, `resw`, `resb` directives
  - `%define`
- **System calls for read & write**
- ~~**Separate compilation, linking & loading**~~
- **Interrupts (general principles)**

LINKING IN ASSEMBLY

The Compilation Process: Major Steps

- **Lexical Analysis**
  - ◇ Converts source code to a token stream
- **Parsing**
  - ◇ Construct a parse tree from the token stream
- **Code Generation**
  - ◇ Produce native assembly language code from parse tree
- **Assembling**
  - ◇ Produce machine language code from assembly language source
- **Linking & Loading**
  - ◇ Resolve external references
  - ◇ Assign addresses to code and data sections

LEXICAL ANALYSIS

Lexical Analysis

- Groups together characters into "tokens"
- recognizes keywords, identifiers, constants, ...
- strips out comments, white space, ...
- Unix tool for lexical analysis: **lex**

```
if ( x + y <= 74.2 ) {
   a = x + 7 ;
else {
   printf ( "Out of bounds!\n" ) ;
}
```

PARSING

Parsing

- Uses context-free grammar (a.k.a. Backus-Naur Form) for the language to construct a parse tree.

A simple grammar:

```
E -> E + T
E -> E - T
E -> T
T -> T * F
T -> T / F
T -> F
F -> <id>
F -> <const>
F -> ( E )
```

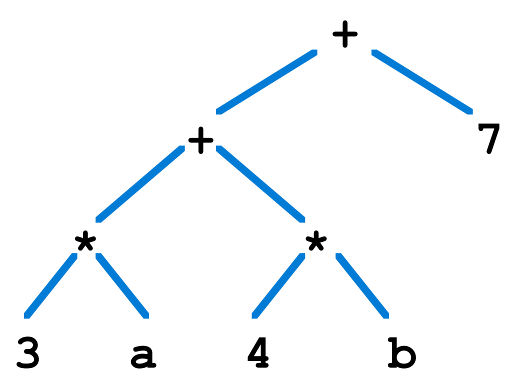
Deriving 3 * a + 4 * b +7:

```
E -> E + T
  -> E + T + T
  -> T + T + T
  -> T * F + T + T
  -> F * F + T + T
  -> 3 * F + T + T
  -> 3 * a + T + T
  -> 3 * a + T * F + T
  -> 3 * a + F * F + T
  -> 3 * a + 4 * F + T
  -> 3 * a + 4 * b + T
  -> 3 * a + 4 * b + 7
```

Parse Trees

- Constructing a parse tree is essentially the reverse of the derivation process
- Unix tool: **yacc** (yet another compiler compiler)

Parse tree for 3 * a + 4 * b +7:



CODE GENERATION

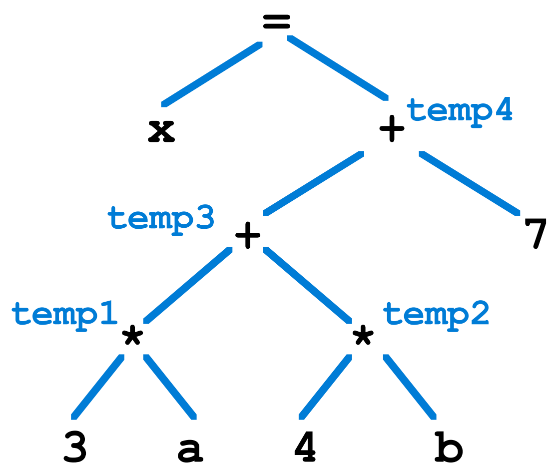
Code Generation

- Produce "intermediate" code from parse tree.
- Produce native assembly language code from intermediate code.
- Code optimization may be used in both steps.

Code Generation Example 1

- Use EAX to perform +, *, ...
- Store result in temporary location

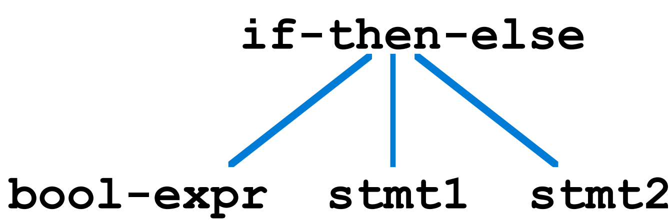
Parse tree for x = 3 * a + 4 * b +7:




```
mov     eax, 3
imul    eax, [a]
mov     [temp1], eax
mov     eax, 4
imul    eax, [b]
mov     [temp2], eax
mov     eax, [temp1]
add     eax, [temp2]
mov     [temp3], eax
mov     eax, [temp3]
add     eax, 7
mov     [temp4], eax
mov     eax, [temp4]
mov     [x], eax
```

Code Generation Example 2

Parse tree for if-then-else statements




```
bool_expr:
         .
         .
         .
        mov    eax, [temp1]
        cmp    eax, 0
        je     stmt2

stmt1:
         .
         .
         .
        jmp    end_if

stmt2:
         .
         .
         .

end_if
```

ASSEMBLING

Assembling

- Line-by-line translation of assembly language mnemonics to machine code
- two passes needed to resolve forward jumps

```
                                  ; File: add2.asm
                                  ;
                                  ; Various addressing modes with the add operation.
                                  ;

                                          section .data

00000000 2A000000                 x:      dd      42                ; 4-byte word


                                          section .text
                                          global _start

00000000 90                       _start: nop

                                          ; initialize

00000001 B811000000               start:  mov     eax, 17           ; eax := 17
00000006 BB[00000000]                     mov     ebx, x            ; ebx := address of x
0000000B B909000000                       mov     ecx, 9            ; ecx := 9

00000010 0503000000                       add     eax, 3            ; add immediate
00000015 01C8                             add     eax, ecx          ; add 32-bit registers
00000017 6601C8                           add     ax, cx            ; add 16-bit registers
0000001A 0305[00000000]                   add     eax, [x]          ; add memory
00000020 0303                             add     eax, [ebx]        ; add register indirect
00000022 8105[00000000]0500-              add     [x], dword 5      ; add immediate to mem
0000002A 0000
0000002C 0105[00000000]                   add     [x], eax          ; add register to mem

                                  ; these two are not allowed (commented out):
                                  ;       add     [x], [x]          ; add mem to mem
                                  ;       add     [x], [ebx]        ; add reg indirect to mem

```

ADD—Add

Opcode	Instruction	Description
04 *ib*	ADD AL,*imm8*	Add *imm8* to AL
05 *iw*	ADD AX,*imm16*	Add *imm16* to AX
05 *id*	ADD EAX,*imm32*	Add *imm32* to EAX
80 /0 *ib*	ADD *r/m8,imm8*	Add *imm8* to *r/m8*
81 /0 *iw*	ADD *r/m16,imm16*	Add *imm16* to *r/m16*
81 /0 *id*	ADD *r/m32,imm32*	Add *imm32* to *r/m32*
83 /0 *ib*	ADD *r/m16,imm8*	Add sign-extended *imm8* to *r/m16*
83 /0 *ib*	ADD *r/m32,imm8*	Add sign-extended *imm8* to *r/m32*
00 /*r*	ADD *r/m8,r8*	Add *r8* to *r/m8*
01 /*r*	ADD *r/m16,r16*	Add *r16* to *r/m16*
01 /*r*	ADD *r/m32,r32*	Add r32 to *r/m32*
02 /*r*	ADD *r8,r/m8*	Add *r/m8* to *r8*
03 /*r*	ADD *r16,r/m16*	Add *r/m16* to *r16*
03 /*r*	ADD *r32,r/m32*	Add *r/m32* to *r32*

Description

Adds the first operand (destination operand) and the second operand (source operand) and stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The ADD instruction performs integer addition. It evaluates the result for both signed and unsigned integer operands and sets the OF and CF flags to indicate a carry (overflow) in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically.

Operation

```
DEST    DEST + SRC;
```

Flags Affected

The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

Table 2-2. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) /digit (Opcode) REG =			AL AX EAX MM0 XMM0 0 000	CL CX ECX MM1 XMM1 1 001	DL DX EDX MM2 XMM2 2 010	BL BX EBX MM3 XMM3 3 011	AH SP ESP MM4 XMM4 4 100	CH BP EBP MM5 XMM5 5 101	DH SI ESI MM6 XMM6 6 110	BH DI EDI MM7 XMM7 7 111
Effective Address	**Mod**	**R/M**	**Value of ModR/M Byte (in Hexadecimal)**							
[EAX]	00	000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]		010	02	0A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	1B	23	2B	33	3B
[--][--][1]		100	04	0C	14	1C	24	2C	34	3C
disp32[2]		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
disp8[EAX][3]	01	000	40	48	50	58	60	68	70	78
disp8[ECX]		001	41	49	51	59	61	69	71	79
disp8[EDX]		010	42	4A	52	5A	62	6A	72	7A
disp8[EBX];		011	43	4B	53	5B	63	6B	73	7B
disp8[--][--]		100	44	4C	54	5C	64	6C	74	7C
disp8[EBP]		101	45	4D	55	5D	65	6D	75	7D
disp8[ESI]		110	46	4E	56	5E	66	6E	76	7E
disp8[EDI]		111	47	4F	57	5F	67	6F	77	7F
disp32[EAX]	10	000	80	88	90	98	A0	A8	B0	B8
disp32[ECX]		001	81	89	91	99	A1	A9	B1	B9
disp32[EDX]		010	82	8A	92	9A	A2	AA	B2	BA
disp32[EBX]		011	83	8B	93	9B	A3	AB	B3	BB
disp32[--][--]		100	84	8C	94	9C	A4	AC	B4	BC
disp32[EBP]		101	85	8D	95	9D	A5	AD	B5	BD
disp32[ESI]		110	86	8E	96	9E	A6	AE	B6	BE
disp32[EDI]		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL/MM0/XMM0	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL/MM/XMM1		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL/MM2/XMM2		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL/MM3/XMM3		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH/MM4/XMM4		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH/MM5/XMM5		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH/MM6/XMM6		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH/MM7/XMM7		111	C7	CF	D7	DF	E7	EF	F7	FF

NOTES:

1. The [--][--] nomenclature means a SIB follows the ModR/M byte.
2. The disp32 nomenclature denotes a 32-bit displacement following the SIB byte, to be added to the index.
3. The disp8 nomenclature denotes an 8-bit displacement following the SIB byte, to be sign-extended and added to the index.

Table 2-1. 16-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) /digit (Opcode) REG =			AL AX EAX MM0 XMM0 0 000	CL CX ECX MM1 XMM1 1 001	DL DX EDX MM2 XMM2 2 010	BL BX EBX MM3 XMM3 3 011	AH SP ESP MM4 XMM4 4 100	CH BP[1] EBP MM5 XMM5 5 101	DH SI ESI MM6 XMM6 6 110	BH DI EDI MM7 XMM7 7 111
Effective Address	**Mod**	**R/M**	**Value of ModR/M Byte (in Hexadecimal)**							
[BX+SI]	00	000	00	08	10	18	20	28	30	38
[BX+DI]		001	01	09	11	19	21	29	31	39
[BP+SI]		010	02	0A	12	1A	22	2A	32	3A
[BP+DI]		011	03	0B	13	1B	23	2B	33	3B
[SI]		100	04	0C	14	1C	24	2C	34	3C
[DI]		101	05	0D	15	1D	25	2D	35	3D
disp16[2]		110	06	0E	16	1E	26	2E	36	3E
[BX]		111	07	0F	17	1F	27	2F	37	3F
[BX+SI]+disp8[3]	01	000	40	48	50	58	60	68	70	78
[BX+DI]+disp8		001	41	49	51	59	61	69	71	79
[BP+SI]+disp8		010	42	4A	52	5A	62	6A	72	7A
[BP+DI]+disp8		011	43	4B	53	5B	63	6B	73	7B
[SI]+disp8		100	44	4C	54	5C	64	6C	74	7C
[DI]+disp8		101	45	4D	55	5D	65	6D	75	7D
[BP]+disp8		110	46	4E	56	5E	66	6E	76	7E
[BX]+disp8		111	47	4F	57	5F	67	6F	77	7F
[BX+SI]+disp16	10	000	80	88	90	98	A0	A8	B0	B8
[BX+DI]+disp16		001	81	89	91	99	A1	A9	B1	B9
[BP+SI]+disp16		010	82	8A	92	9A	A2	AA	B2	BA
[BP+DI]+disp16		011	83	8B	93	9B	A3	AB	B3	BB
[SI]+disp16		100	84	8C	94	9C	A4	AC	B4	BC
[DI]+disp16		101	85	8D	95	9D	A5	AD	B5	BD
[BP]+disp16		110	86	8E	96	9E	A6	AE	B6	BE
[BX]+disp16		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL/MM0/XMM0	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL/MM1/XMM1		001	C1	C9	D1	D9	EQ	E9	F1	F9
EDX/DX/DL/MM2/XMM2		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL/MM3/XMM3		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AHMM4/XMM4		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH/MM5/XMM5		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH/MM6/XMM6		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH/MM7/XMM7		111	C7	CF	D7	DF	E7	EF	F7	FF

NOTES:

1. The default segment register is SS for the effective addresses containing a BP index, DS for other effective addresses.
2. The “disp16” nomenclature denotes a 16-bit displacement following the ModR/M byte, to be added to the index.
3. The “disp8” nomenclature denotes an 8-bit displacement following the ModR/M byte, to be sign-extended and added to the index.

LINKING & LOADING

Linking & Loading

- Linking resolves external references to data and code, including calls to library functions.
- References are often raw addresses without type.
- Loading assigns addresses to data & code sections.
- The loader must patch every absolute memory reference in the code with the assigned address:

```
MOV     EAX, [x]        ; value of x is patched by the loader
```

- In UNIX, `ld` performs both linking & loading.

```
; File: twopass.asm
;
; Demonstrating a two-pass assembler

        section .data
x:      db      87h
y:      dw      1492h
zalias  equ $
z:      dd      17762001h

calc equ (x-y)*2
x4 equ x+1

        section .text
        global _start

_start: mov     eax, [zalias]
        mov     bx, [y]
        mov     cx, [x4]
        cmp     bx, cx
        jne     error

OK:     add     ax, bx
        mov     [x], al
        mov     ebx, 0          ; 0=normal exit

done:   mov     eax, 1          ; syscall number for exit
        int     080h

error:  mov     ebx, 17         ; abnormal exit
        jmp     done
```

```
                          ; File: twopass.asm
                          ;
                          ; Demonstrating a two-pass assembler

                                  section .data
00000000 87               x:      db      87h
00000001 9214             y:      dw      1492h
                          zalias  equ $
00000003 01207617         z:      dd      17762001h

                          calc equ (x-y)*2
                          x4 equ x+1

                                  section .text
                                  global _start

00000000 A1[03000000]     _start: mov     eax, [zalias]
00000005 668B1D[01000000]         mov     bx, [y]
0000000C 668B0D[01000000]         mov     cx, [x4]
00000013 6639CB                   cmp     bx, cx
00000016 7514                     jne     error

00000018 6601D8           OK:     add     ax, bx
0000001B A2[00000000]             mov     [x], al
00000020 BB00000000               mov     ebx, 0          ; 0=normal exit

00000025 B801000000       done:   mov     eax, 1          ; syscall number for exit
0000002A CD80                     int     080h

0000002C BB11000000       error:  mov     ebx, 17         ; abnormal exit
00000031 E9EFFFFFFF               jmp     done
```

```
linux3% nasm -f elf -l twopass.lst twopass.asm
linux3% ld twopass.o

linux3% a.out  ; echo $?
0

linux3% objdump -t twopass.o

twopass.o:     file format elf32-i386

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 twopass.asm
00000000 l    d  *ABS*  00000000
00000000 l    d  .data  00000000
00000000 l    d  .text  00000000
00000000 l       .data  00000000 x
00000001 l       .data  00000000 y
00000003 l       .data  00000000 zalias
00000003 l       .data  00000000 z
fffffffe l       *ABS*  00000000 calc
00000001 l       .data  00000000 x4
00000018 l       .text  00000000 OK
00000025 l       .text  00000000 done
0000002c l       .text  00000000 error
00000000 g       .text  00000000 _start
```

```
linux3% objdump -t a.out

a.out:     file format elf32-i386

SYMBOL TABLE:
08048080 l    d  .text  00000000
080490b8 l    d  .data  00000000
080490bf l    d  .bss   00000000
00000000 l    d  .comment       00000000
00000000 l    d  *ABS*  00000000
00000000 l    d  *ABS*  00000000
00000000 l    d  *ABS*  00000000
00000000 l    df *ABS*  00000000 twopass.asm
080490b8 l       .data  00000000 x
080490b9 l       .data  00000000 y
080490bb l       .data  00000000 zalias
080490bb l       .data  00000000 z
fffffffe l       *ABS*  00000000 calc
080490b9 l       .data  00000000 x4
08048098 l       .text  00000000 OK
080480a5 l       .text  00000000 done
080480ac l       .text  00000000 error
080480b6 g     O *ABS*  00000000 _etext
08048080 g       .text  00000000 _start
080490bf g     O *ABS*  00000000 __bss_start
080490bf g     O *ABS*  00000000 _edata
080490c0 g     O *ABS*  00000000 _end


linux3% objdump -h a.out

a.out:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000036  08048080  08048080  00000080  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000007  080490b8  080490b8  000000b8  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000001  080490bf  080490bf  000000bf  2**0
                  CONTENTS
  3 .comment      0000001c  00000000  00000000  000000c0  2**0
                  CONTENTS, READONLY
linux3%
```

```
linux3% objdump -d a.out

a.out:     file format elf32-i386

Disassembly of section .text:

08048080 <_start>:
 8048080:       a1 bb 90 04 08          mov    0x80490bb,%eax
 8048085:       66 8b 1d b9 90 04 08    mov    0x80490b9,%bx
 804808c:       66 8b 0d b9 90 04 08    mov    0x80490b9,%cx
 8048093:       66 39 cb                cmp    %cx,%bx
 8048096:       75 14                   jne    80480ac <error>

08048098 <OK>:
 8048098:       66 01 d8                add    %bx,%ax
 804809b:       a2 b8 90 04 08          mov    %al,0x80490b8
 80480a0:       bb 00 00 00 00          mov    $0x0,%ebx

080480a5 <done>:
 80480a5:       b8 01 00 00 00          mov    $0x1,%eax
 80480aa:       cd 80                   int    $0x80

080480ac <error>:
 80480ac:       bb 11 00 00 00          mov    $0x11,%ebx
 80480b1:       e9 ef ff ff ff          jmp    80480a5 <done>
```

```
; File: sep1.asm
;
; File 1 for separate compilation example

global gvar1, _start
extern gvar2, add_these

        section .data
foo:    db      12h
gvar1:  dd      17h
lvar1:  dd      42h

        section .text
_start: mov     eax, [gvar1]
        mov     ebx, [gvar2]
        mov     ecx, [lvar1]

        call    add_these        ; gvar1 := eax+ebx+ecx
        mov     ebx, [gvar1]     ; store in return code
        mov     eax, 1           ; sycall number for exit
        int     080h             ; bye-bye
```

```
; File: sep2.asm
;
; File 2 for separate compilation example

global gvar2, add_these
extern gvar1

        section .data
bar:    dw      07h
gvar2:  dd      03h
lvar1:  dd      02h      ; same name as other lvar1, OK

        section .text
add_these:                              ; no regs altered!
        mov     [gvar1], dword 0        ; clear destination
        add     [gvar1], eax
        add     [gvar1], ebx
        add     [gvar1], ecx
        ret
```

```
                               ; File: sep1.asm
                               ;
                               ; File 1 for separate compilation example

                               global gvar1, _start
                               extern gvar2, add_these

                                       section .data

00000000 12                    foo:    db      12h
00000001 17000000              gvar1:  dd      17h
00000005 42000000              lvar1:  dd      42h

                                       section .text
00000000 A1[01000000]          _start: mov     eax, [gvar1]
00000005 8B1D[00000000]                mov     ebx, [gvar2]
0000000B 8B0D[05000000]                mov     ecx, [lvar1]

00000011 E8(00000000)                  call    add_these       ; gvar1 := eax+ebx+ecx
00000016 8B1D[01000000]                mov     ebx, [gvar1]    ; store in return code
0000001C B801000000                    mov     eax, 1          ; sycall number for exit
00000021 CD80                          int     080h            ; bye-bye
```

```
                                ; File: sep2.asm
                                ;
                                ; File 2 for separate compilation example

                                global gvar2, add_these
                                extern gvar1

                                 section .data

00000000 0700                   bar:  dw   07h
00000002 03000000               gvar2:     dd   03h
00000006 02000000               lvar1:     dd   02h ; same name as other lvar1, OK

                                 section .text
                                add_these:                 ; no regs altered!
00000000 C705[00000000]0000-     mov [gvar1], dword 0 ; clear destination
00000008 0000
0000000A 0105[00000000]          add [gvar1], eax
00000010 011D[00000000]          add [gvar1], ebx
00000016 010D[00000000]          add [gvar1], ecx
0000001C C3                      ret
```

```
linux3% nasm -f elf -l sep1.lst sep1.asm
linux3% nasm -f elf -l spe2.lst sep2.asm
linux3% ld sep1.o sep2.o
linux3% a.out
linux3% echo $?
92
linux3%
```

```
linux3% objdump -h sep1.o

sep1.o:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .data         00000009  00000000  00000000  00000180  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  1 .text         00000023  00000000  00000000  00000190  2**4
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  2 .comment      0000001c  00000000  00000000  000001c0  2**0
                  CONTENTS, READONLY
linux3%
```

```
linux3% objdump -t sep1.o

sep1.o:     file format elf32-i386

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 sep1.asm
00000000 l    d  *ABS*  00000000
00000000 l    d  .data  00000000
00000000 l    d  .text  00000000
00000000 l       .data  00000000 foo
00000005 l       .data  00000000 lvar1
00000000         *UND*  00000000 gvar2
00000000         *UND*  00000000 add_these
00000001 g       .data  00000000 gvar1
00000000 g       .text  00000000 _start
```

```
linux3% objdump -h sep2.o

sep2.o:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .data         0000000a  00000000  00000000  00000180  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  1 .text         0000001d  00000000  00000000  00000190  2**4
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  2 .comment      0000001c  00000000  00000000  000001b0  2**0
                  CONTENTS, READONLY
linux3%
linux3% objdump -t sep2.o

sep2.o:     file format elf32-i386

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 sep2.asm
00000000 l    d  *ABS*  00000000
00000000 l    d  .data  00000000
00000000 l    d  .text  00000000
00000000 l       .data  00000000 bar
00000006 l       .data  00000000 lvar1
00000000         *UND*  00000000 gvar1
00000002 g       .data  00000000 gvar2
00000000 g       .text  00000000 add_these
```

```
linux3% objdump -h a.out

a.out:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000004d  08048080  08048080  00000080  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000016  080490d0  080490d0  000000d0  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000002  080490e6  080490e6  000000e6  2**0
                  CONTENTS
  3 .comment      00000038  00000000  00000000  000000e8  2**0
                  CONTENTS, READONLY
linux3% objdump -t a.out

a.out:     file format elf32-i386

SYMBOL TABLE:
08048080 l    d  .text  00000000
080490d0 l    d  .data  00000000
080490e6 l    d  .bss   00000000
00000000 l    d  .comment       00000000
00000000 l    d  *ABS*  00000000
00000000 l    d  *ABS*  00000000
00000000 l    d  *ABS*  00000000
00000000 l    df *ABS*  00000000 sep1.asm
080490d0 l       .data  00000000 foo
080490d5 l       .data  00000000 lvar1
00000000 l    df *ABS*  00000000 sep2.asm
080490dc l       .data  00000000 bar
080490e2 l       .data  00000000 lvar1
080480cd g     O *ABS*  00000000 _etext
080480b0 g       .text  00000000 add_these
08048080 g       .text  00000000 _start
080490de g       .data  00000000 gvar2
080490e6 g     O *ABS*  00000000 __bss_start
080490d1 g       .data  00000000 gvar1
080490e6 g     O *ABS*  00000000 _edata
080490e8 g     O *ABS*  00000000 _end
```

```
linux3% objdump -d a.out

a.out:     file format elf32-i386

Disassembly of section .text:

08048080 <_start>:
 8048080:       a1 d1 90 04 08          mov    0x80490d1,%eax
 8048085:       8b 1d de 90 04 08       mov    0x80490de,%ebx
 804808b:       8b 0d d5 90 04 08       mov    0x80490d5,%ecx
 8048091:       e8 1a 00 00 00          call   80480b0 <add_these>
 8048096:       8b 1d d1 90 04 08       mov    0x80490d1,%ebx
 804809c:       b8 01 00 00 00          mov    $0x1,%eax
 80480a1:       cd 80                   int    $0x80
 80480a3:       90                      nop
 80480a4:       90                      nop
 80480a5:       90                      nop
 80480a6:       90                      nop
 80480a7:       90                      nop
 80480a8:       90                      nop
 80480a9:       90                      nop
 80480aa:       90                      nop
 80480ab:       90                      nop
 80480ac:       90                      nop
 80480ad:       90                      nop
 80480ae:       90                      nop
 80480af:       90                      nop

080480b0 <add_these>:
 80480b0:       c7 05 d1 90 04 08 00    movl   $0x0,0x80490d1
 80480b7:       00 00 00
 80480ba:       01 05 d1 90 04 08       add    %eax,0x80490d1
 80480c0:       01 1d d1 90 04 08       add    %ebx,0x80490d1
 80480c6:       01 0d d1 90 04 08       add    %ecx,0x80490d1
 80480cc:       c3                      ret
linux3%
```

```
linux3% objdump -s a.out

a.out:     file format elf32-i386

Contents of section .text:
 8048080 a1d19004 088b1dde 9004088b 0dd59004  ................
 8048090 08e81a00 00008b1d d1900408 b8010000  ................
 80480a0 00cd8090 90909090 90909090 90909090  ................
 80480b0 c705d190 04080000 00000105 d1900408  ................
 80480c0 011dd190 0408010d d1900408 c3        .............
Contents of section .data:
 80490d0 12170000 00420000 00000000 07000300  .....B..........
 80490e0 00000200 0000                        ......
Contents of section .bss:
 80490e6 0000                                 ..
Contents of section .comment:
 0000 00546865 204e6574 77696465 20417373  .The Netwide Ass
 0010 656d626c 65722030 2e393800 00546865  embler 0.98..The
 0020 204e6574 77696465 20417373 656d626c   Netwide Assembl
 0030 65722030 2e393800                    er 0.98.
linux3% exit
```

Shows relative jumps
to labels in another file.

```
; File: sep3.asm
;
; File 3 for separate compilation example

extern _start, add_these

        section .data

lvar1:  dd      03h     ; same name as other lvar1, OK

        section .text
test3:                          ; no regs altered!
        cmp     [lvar1], dword 7
        jne     _start
        jmp     add_these
```

```
linuxserver1% nasm -f elf sep3.asm

linuxserver1% objdump -t sep3.o

sep3.o:     file format elf32-i386

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 sep3.asm
00000000 l    d  *ABS*  00000000
00000000 l    d  .data  00000000
00000000 l    d  .text  00000000
00000000 l       .data  00000000 lvar1
00000000 l       .text  00000000 test3
00000000         *UND*  00000000 _start
00000000         *UND*  00000000 add_these


linuxserver1% ld sep1.o sep2.o sep3.o
```

```
linuxserver1% objdump -t a.out

a.out:     file format elf32-i386

SYMBOL TABLE:
08048080 l    d  .text  00000000
080490e8 l    d  .data  00000000
080490fc l    d  .bss   00000000
00000000 l    d  .comment       00000000
00000000 l    d  *ABS*  00000000
00000000 l    d  *ABS*  00000000
00000000 l    d  *ABS*  00000000
00000000 l    df *ABS*  00000000 sep1.asm
080490ec l       .data  00000000 lvar1
00000000 l    df *ABS*  00000000 sep2.asm
080490f4 l       .data  00000000 lvar1
00000000 l    df *ABS*  00000000 sep3.asm
080490f8 l       .data  00000000 lvar1
080480d0 l       .text  00000000 test3
080480b0 g       .text  00000000 add_these
08048080 g       .text  00000000 _start
080490f0 g       .data  00000000 gvar2
080490fc g       *ABS*  00000000 __bss_start
080490e8 g       .data  00000000 gvar1
080490fc g       *ABS*  00000000 _edata
080490fc g       *ABS*  00000000 _end
```

```
linuxserver1% objdump -d a.out
a.out:     file format elf32-i386

Disassembly of section .text:

08048080 <_start>:
 8048080:       a1 e8 90 04 08          mov    0x80490e8,%eax
 8048085:       8b 1d f0 90 04 08       mov    0x80490f0,%ebx
 804808b:       8b 0d ec 90 04 08       mov    0x80490ec,%ecx
 8048091:       e8 1a 00 00 00          call   80480b0 <add_these>
 8048096:       8b 1d e8 90 04 08       mov    0x80490e8,%ebx
 804809c:       b8 01 00 00 00          mov    $0x1,%eax
 80480a1:       cd 80                   int    $0x80

080480b0 <add_these>:
 80480b0:       c7 05 e8 90 04 08 00    movl   $0x0,0x80490e8
 80480b7:       00 00 00
 80480ba:       01 05 e8 90 04 08       add    %eax,0x80490e8
 80480c0:       01 1d e8 90 04 08       add    %ebx,0x80490e8
 80480c6:       01 0d e8 90 04 08       add    %ecx,0x80490e8
 80480cc:       c3                      ret

080480d0 <test3>:
 80480d0:       81 3d f8 90 04 08 07    cmpl   $0x7,0x80490f8
 80480d7:       00 00 00
 80480da:       0f 85 a0 ff ff ff       jne    8048080 <_start>
 80480e0:       e9 cb ff ff ff          jmp    80480b0 <add_these>
```

FUNCTIONS & SEPARATE COMPILATION IN C

C Parameter Passing Notes

- We'll say *formal parameter* vs *actual parameter*.
  - Formal parameters are place holders in function definition.
  - Actual parameters (aka arguments) actually have a value.
- In C, all parameters are passed by value.
- Parameter passing by reference is simulated by passing the *address* of the variable.

```
scanf("%d", &n) ;
```

- Array names represent the address of the array. In effect, arrays are passed by reference.

```
int UpdateArray (int A[], int n) {
   A[0] += 5 ;
   ...
```

A Simple C Program

```
#include <stdio.h>
typedef double Radius;
#define PI 3.1415

double circleArea( Radius radius ) {
   return PI * radius * radius ;
}

double calcCircumference( Radius radius ) {
   return 2 * PI * radius ;
}

int main() {
   Radius radius = 4.5;
   double area = circleArea( radius );
   double circumference = calcCircumference( radius );

   printf ("Area = %10.2f, Circumference = %10.2f\n",
      area, circumference);

   return 0;
}
```

Adapted from Dennis Frey CMSC 313 Spring 2011

Separate Compilation: Why?

- **Keeps files small.**
- **Different people can work on different parts of the program.**
- **Easier to find functions.**
- **Keeps large program logically organized.**
- **Do not have to re-compile entire program when changes are made to a small portion.**
- **Parts of the program (e.g., code for a data structure) may be reusable in other programs.**

Problem: need a mechanism for external references.

circleUtils.h

```
/* circleUtils.h*/

/* #includes required by the prototypes, if any */


/* typedefs and #defines */

typedef double Radius;


/* function prototypes */

double circleArea( Radius radius );

double calcCircumference( Radius radius );
```

circleUtils.c

```
/* circleUtils.c */

#include "circleUtils.h"
#define PI 3.1415

/* Function implementations */

double circleArea( Radius radius ) {

   return ( PI * radius * radius );
}

double calcCircumference( Radius radius ) {

   return (2 * PI * radius );
}
```

Adapted from Dennis Frey CMSC 313 Spring 2011

main program

```
/* sample.c */
#include <stdio.h>
#include "circleUtils.h"

int main( ) {
   Radius radius = 4.5;
   double area, circumference ;

   area = circleArea( radius );
   circumference = calcCircumference( radius );

   printf (“Area = %lf, Circumference = %lf\n”,
        area, circumference);

   return 0;
}
```

Header Files

- Header files should contain
  - function prototypes
  - type definitions
  - #define constants
  - extern declarations for global variables
  - other #includes
- Header files should end with .h
- System header files #included with < >

```
#include <stdio.h>
```

- Your own header files #included with " "

```
#include "circleUtils.h"
```

- Header files are expected to include all other header files needed to work with implemented functions.

Guarding Header Files

- Header files should not be included multiple times.
- multiple declaration of function prototypes: OK
- multiple type definition: BAD
- multiple `#include` can lead to loops where a `.h` file includes itself.

- Solution:

```
#ifndef _UNIQUE_VAR_NAME_
#define _UNIQUE_VAR_NAME_

...
#endif
```

Guarded circleUtils.h

```
#ifndef CIRCLEUTIL_H
#define CIRCLEUTIL_H

/* circleUtils.h*/

/* #includes required by the prototypes, if any */

/* typedefs and #defines */

typedef double Radius;

/* function prototypes */

double circleArea( Radius radius );

double calcCircumference( Radius radius );

#endif
```

Compiling and linking

- **How to compile:**

```
gcc -c -Wall circleUtils.c
gcc -c -Wall sample.c
gcc -Wall -o sample sample.o circleutils.o
```

- **Or**

```
gcc -Wall -o sample sample.c circleUtils.c
```

Compiler vs linker

- **Compiler: translates one .c file into a .o file**
  - **Verifies that all functions are being called correctly**
  - **Verifies that all variables exist**
  - **Verifies language syntax**

- **Linker: combines .o files and C libraries into executable file**
  - **"Finds" functions called by one .c/.o file, but defined in another E.g. printf( ), scanf( ).**
  - **"Finds" global variables used by one .c/.o file, but defined in another (more on this soon)**

- **`gcc` uses `ld` to link & load**
  - **Easier to invoke `ld` through `gcc`**

Linking with C libraries

- By default, the standard C library which includes printf, scanf and char and string functions is always linked with your program.

- Other libraries must be explicitly linked with your code.

- Typical C libraries have the form `libxxx.a`.
  - Standard C library: `libc.a`.
  - Math library:`libm.a`.

- Use the `-l` flag and the `xxx` part of the library name to link.

```
gcc -Wall -o sample sample.c circleUtils.c -lm
```

Project Organization

- main( ) is generally defined in its own .c file and generally just calls helper functions
  - E.g. project1.c
- Project-specific helper functions may be in the same .c file as main( )
  - main( ) comes first
  - Helper function order that makes sense to you
- Reusable functions in their own .c file
  - Group related functions in the same file
  - E.g. circleUtils.c
- Prototypes, typedefs, #defines, etc. for reusable function in a .h file
  - Same file root name as the .c file. E.g. circleUtils.h

SCOPE & LIFETIME

Variable Scope and Lifetime

- The scope of a variable refers to that part of a program that may refer to the variable.

- The lifetime of a variable refers to the time in which a variable occupies a place in memory.

- The scope and lifetime of a variable are determined by how and where the variable is defined.

static and extern

- In C/C++, the keyword static is overloaded.
  - A static local variable has lifetime = duration of program.
  - A static global variable has file scope
  - A static function has file scope
- extern is means that the variable is defined in another file.

```
extern int other_variable ;
```

- an extern declaration is an example of a declaration that is not a definition. (Another example is a function prototype.)

Adapted from Dennis Frey CMSC 313 Spring 2011

NEXT TIME

- **Pointers**

References

- Some figures and diagrams from *IA-32 Intel Architecture Software Developer's Manual, Vols 1-3*

  <http://developer.intel.com/design/Pentium4/manuals/>