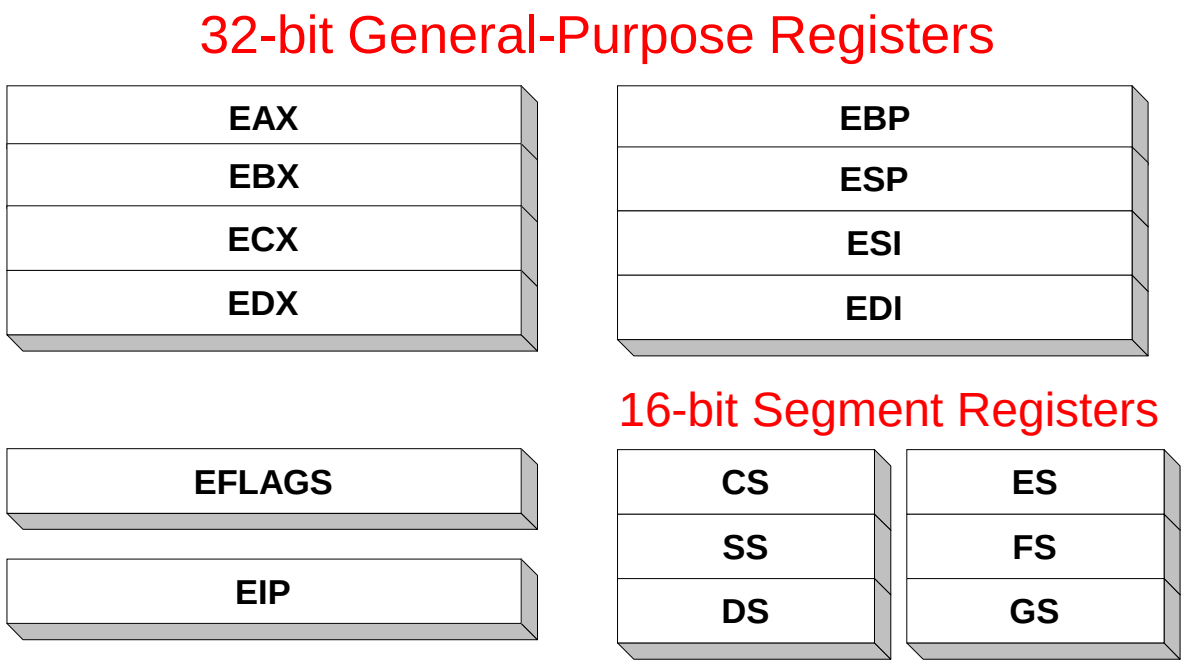
IA-32 Processor

Programmer's perspective

32-bit programming

Basic Program Execution Registers

- Registers are high speed memory inside the CPU
  - Eight 32-bit general-purpose registers
  - Six 16-bit segment registers
  - Processor Status Flags (EFLAGS) and Instruction Pointer (EIP)

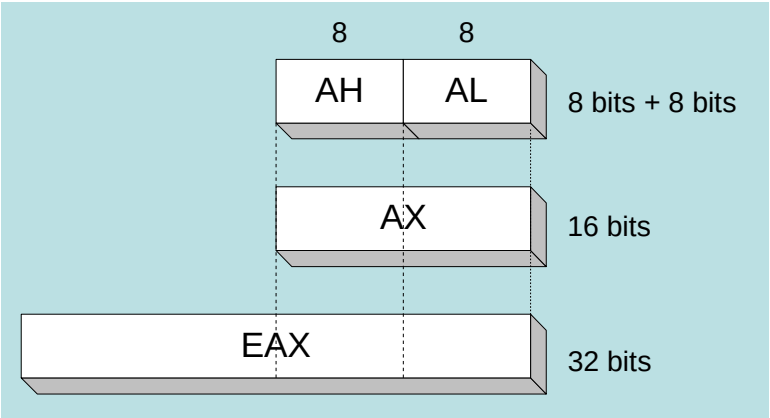


General-Purpose Registers

- Used primarily for arithmetic and data movement
  - `mov eax, 10` move constant 10 into register eax
- Specialized uses of Registers
  - EAX – Accumulator register
    - Automatically used by multiplication and division instructions
  - ECX – Counter register
    - Automatically used by LOOP instructions
  - ESP – Stack Pointer register
    - Used by PUSH and POP instructions, points to top of stack
  - ESI and EDI – Source Index and Destination Index register
    - Used by string instructions
  - EBP – Base Pointer register
    - Used to reference parameters and local variables on the stack

Accessing Parts of Registers

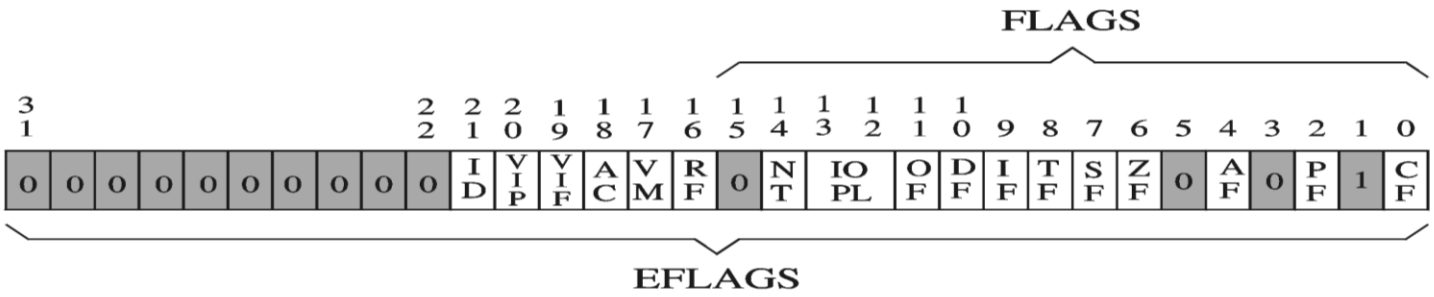
- EAX, EBX, ECX, and EDX are 32-bit Extended registers
  - Programmers can access their 16-bit and 8-bit parts
  - Lower 16-bit of EAX is named AX
  - AX is further divided into
    - AL = lower 8 bits
    - AH = upper 8 bits
- ESI, EDI, EBP, ESP have only 16-bit names for lower half



32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

32-bit	16-bit
ESI	SI
EDI	DI
EBP	BP
ESP	SP

EFLAGS Register



Status flags

CF = Carry flag
PF = Parity flag
AF = Auxiliary carry flag
ZF = Zero flag
SF = Sign flag
OF = Overflow flag

Control flags

DF = Direction flag

System flags

TF = Trap flag
IF = Interrupt flag
IOPL = I/O privilege level
NT = Nested task
RF = Resume flag
VM = Virtual 8086 mode
AC = Alignment check
VIF = Virtual interrupt flag
VIP = Virtual interrupt pending
ID = ID flag

- Status Flags
  - Status of arithmetic and logical operations
- Control and System flags
  - Control the CPU operation
- Programs can set and clear individual bits in the EFLAGS register

Status Flags

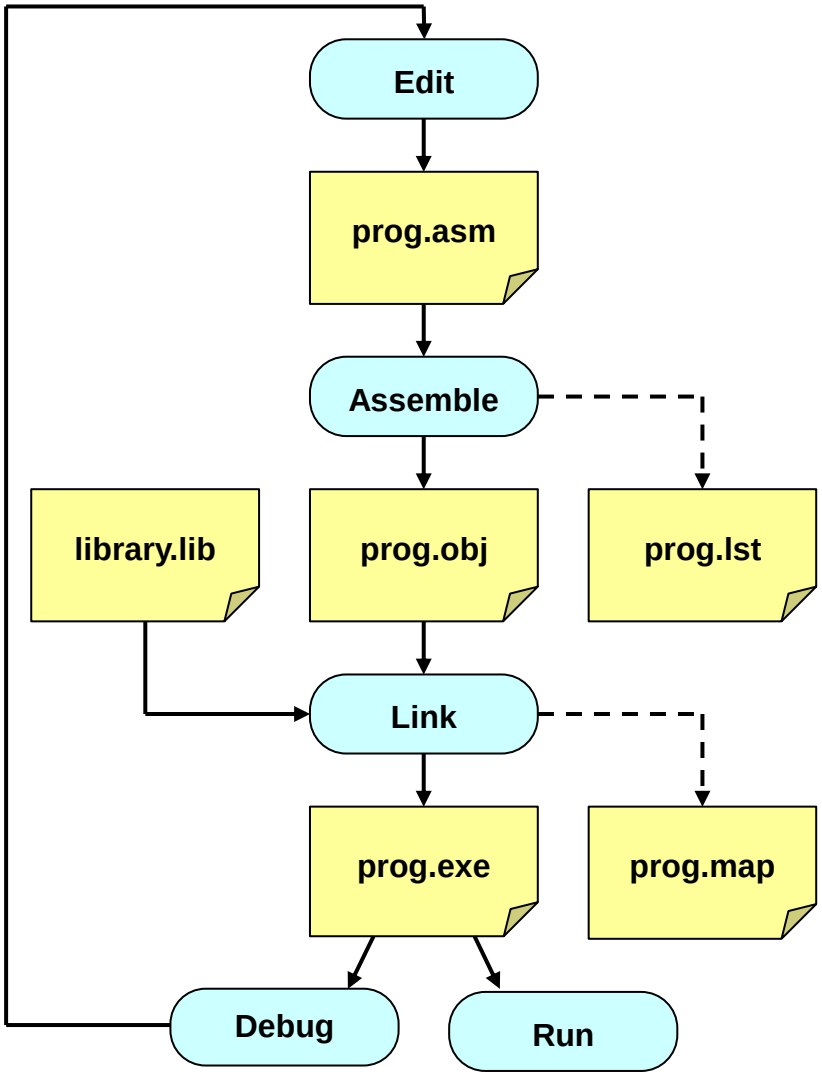
- Carry Flag
  - Set when unsigned arithmetic result is out of range
- Overflow Flag
  - Set when signed arithmetic result is out of range
- Sign Flag
  - Copy of sign bit, set when result is negative
- Zero Flag
  - Set when result is zero
- Auxiliary Carry Flag
  - Set when there is a carry from bit 3 to bit 4
- Parity Flag
  - Set when parity is even
  - Least-significant byte in result contains even number of 1s

Assembly Language Introduction

- Low level language 1:1 correspondence with machine language
- Very simple instructions resembling CPU operations
- Machine dependent language (Not portable)
- Pros
  - Direct hardware control
  - Real time applications
  - System level programs
  - System understanding (behind the scenes)
  - Helps in CS courses e.g computer Architecture, Operating Systems, Compiler construction, system programming etc
- Cons
  - Development slow and difficult to grasp

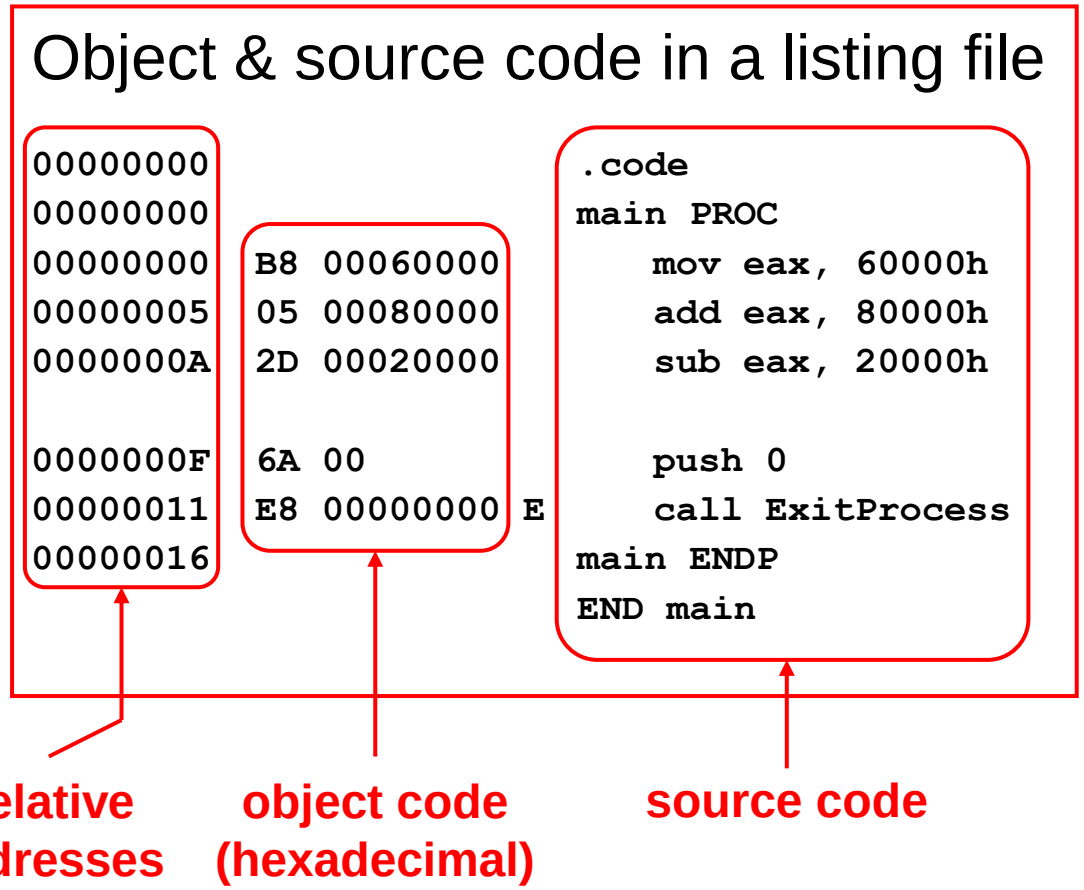
Assembly Program Life Cycle

- Editor
  - Write new (**.asm**) programs
  - Make changes to existing ones
- Assembler: **ML.exe** program
  - Translate (**.asm**) file into object (**.obj**) file in machine language
  - Can produce a listing (**.lst**) file that shows the work of assembler
- Linker: **LINK32.exe** program
  - Combine object (**.obj**) files with link library (**.lib**) files
  - Produce executable (**.exe**) file
  - Can produce optional (**.map**) file



Listing File

- Use it to see how your program is assembled
- Contains
  - Source code
  - Object code
  - Relative addresses
  - Segment names
  - Symbols
    - Variables
    - Procedures
    - Constants



Assembly Language Statements

- Three types of statements in assembly language
  - Typically, one statement on a line

1. Executable Instructions
   - Instructions tell the processor what to do
     - Machine code generated by Assembler
2. Assembler Directives
   - Provide information to the assembler while translating a program
   - Used to define data, select memory model, etc.
   - Non-executable: directives are not part of instruction set
3. Macros
   - Shorthand notation for a group of statements
   - Sequence of instructions, directives, or other macros

How to write an instruction

- Format:

  `[label:]  opcode  [operands]  [;comment]`

- Instruction Label (optional)
  - Gives symbolic name to address of the instruction, must have a colon **:**
  - Used to transfer program execution to a labeled instruction
- opcode
  - Identifies the operation (e.g. MOV, ADD, SUB, JMP, CALL)
- Operands
  - Specify the data required by the operation
  - Executable instructions can have zero to three operands
  - Operands can be registers, memory variables/symbolic names of memory locations, or constants

Types of Operands

- Immediate
  - Constant integer (8, 16, or 32 bits)
  - Constant value is stored within the instruction
- Register
  - Name of a register is specified
  - Register number is encoded within the instruction
- Memory
  - Reference to a location in memory
  - Memory address is encoded within the instruction, or
  - Register holds the address of a memory location

Types of Operations

- Determines instruction type
  - Data Movement (Data Movement Instructions)
  - Arithmetic operations (Arithmetic instructions)
  - Logical operations (Logical Instructions)
  - Control transfer
  - Etc

Instruction Operand Notation

Operand	Description
r8	8-bit general-purpose register: AH, AL, BH, BL, CH, CL, DH, DL
r16	16-bit general-purpose register: AX, BX, CX, DX, SI, DI, SP, BP
r32	32-bit general-purpose register: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
reg	Any general-purpose register
sreg	16-bit segment register: CS, DS, SS, ES, FS, GS
imm	8-, 16-, or 32-bit immediate value
imm8	8-bit immediate byte value
imm16	16-bit immediate word value
imm32	32-bit immediate doubleword value
r/m8	8-bit operand which can be an 8-bit general-purpose register or memory byte
r/m16	16-bit operand which can be a 16-bit general-purpose register or memory word
r/m32	32-bit operand which can be a 32-bit general register or memory doubleword
mem	8-, 16-, or 32-bit memory operand

Instruction Examples

- No operands

```
stc                 ; set carry flag
```

- One operand

```
inc  eax            ; increment register eax
call Clrscr         ; call procedure Clrscr
jmp  L1             ; jump to instruction with label L1
```

- Two operands

```
add  ebx, ecx       ; register ebx = ebx + ecx
sub  var1, 25       ; memory variable var1 = var1 - 25
```

- Three operands

```
imul eax,ebx,5 ; register eax = ebx * 5
```

Program Template you will use

```
INCLUDE Irvine32.inc

.DATA
    ; (insert variables here)

.CODE

main PROC
    ; (insert executable instructions here)
    exit
main ENDP

    ; (insert additional procedures here)

END main
```

Thanks!