The Intel 8088 Architecture and Instruction Set

This lecture describes a subset *of the 8088 architecture and instruction set. While it's not possible to cover all the details of the 8088 in one lecture you should learn enough about the 8088 to be able to:*

- *write simple program in 8088 assembly language including: (1) transfer of 8 and 16-bit data between registers and memory using register, immediate, direct, and register indirect addressing, (2) some essential arithmetic and logic instructions on byte and 16-bit values, (3) stack push/pop, (4) input/output, (5) [un]conditional branches, (6) call/return, (7) interrupt/return, (8) essential pseudo-ops (org, db, dw).*
- *compute a physical address from segment and offset values,*
- *describe response of 8088 to NMI, software (INT) and external (IRQ) interrupts and return from interrupts.*

History

The original Intel 16-bit CPU was the 8086. It was designed to be backwards-compatible at the assembler level with Intel's 8-bit CPU, the 8080. The 8088 is a version of the 8086 with an 8-bit data bus. The 8088 was used in the original IBM PC and its many clones. Later versions of the 8086 include the i386 which extends the data and address registers to 32 bits and includes support for memory protection and virtual memory.

Registers

The 8088 includes has four 16-bit data registers (AX, BX, CX and DX). BX can also be used as an address register for indirect addressing. The most/least significant byte of each register can also be addressed directly (e.g. AL is the LS byte of AX, CH is MS byte of CX).

Three bits in a 16-bit program status word (PSW) are used to indicate whether the result of the previous arithmetic/logical instruction was zero, negative, or generated a carry. An interrupt enable bit controls whether interrupt requests on the IRQ pin are recognized.

The address of the next instruction to be executed is held in a 16-bit instruction pointer (IP) register (the "program counter").

Exercise: How many bytes can be addressed by a 16-bit value?

A 16-bit stack pointer (SP) is used to implement a stack to support subroutine calls and interrupts/exceptions.

There are also three segment registers (CS, DS, SS) which are used to allow the code, data and stack to be located in any three 64 kByte "segments" within a 1 megabyte (20-bit) address space as described below.

Instruction Set

Data Transfer

Transfer of 8 and 16-bit data is done using the MOV instruction. Either the source or destination has to be a register. The other operand can come from another register, from memory, from immediate data (a value encoded in the instruction) or from a memory location "pointed at" by register BX. For example, if COUNT is the label of a memory location the following are possible:

```
; register: move contents of BX to AX
    MOV     AX,BX
; direct: move contents of AX to memory
    MOV     COUNT,AX
; immediate: load CX with the value 240
    MOV     CX,0F0H
; register indirect: move contents of AL
; to memory location in BX
    MOV     [BX],AL
```

16-bit registers can be pushed (SP is first decremented by two and then the value stored at SP) or popped (the value is restored from memory at SP and then SP is incremented by 2). For example:

```
    PUSH    AX      ; push contents of AX
    POP     BX      ; restore into BX
```

I/O Operations

The 8088 has separate I/O and memory address spaces. Values in the I/O space are accessed with IN and OUT instructions. The port address is loaded into DX and the data is read/written to/from AL or AX:

```
    MOV     DX,372H ; load DX with port address
    OUT     DX,AL   ; output byte in AL to port
                    ; 372 (hex)
    IN      AX,DX   ; input word to AX
```

Arithmetic/Logic

Arithmetic and logic instructions can be performed on byte and 16-bit values. The first operand has to be a register and the result is stored in that register.

```
; increment BX by 4
    ADD     BX,4
; subtract 1 from AL
    SUB     AL,1
; increment BX
    INC     BX
; compare (subtract without storing result)
    CMP     AX,MAX
; mask in LS 4 bits of AL
    AND     AL,0FH
; shift AX right
    SHR     AX
; set MS bit of CX
    OR      CX,8000H
; clear AX
    XOR     AX,AX
```

Control Transfer

Conditional jumps transfer control to another address depending on the state of the flags in the PSW. Conditional jumps are restricted to a range of -128 to +127 bytes from the next instruction while unconditional jumps can be to any point within the 64k code segment.

```
; jump if last result was zero (two values equal)
    JZ      skip
; jump on less than
    JL     smaller
; jump if carry set (below)
    JC      neg
; unconditional jump:
    JMP     loop
```

The `CALL` and `RET` instructions call and return from subroutines. The processor pushes IP on the stack during a `CALL` instruction and the contents of IP are popped by the `RET` instructions. For example:

```
    CALL    readchar
    ...
readchar:
    ...
    RET
```

Interrupts and Exceptions

In addition to *interrupts* caused by external events (such as an IRQ signal), certain instructions such as a dividing by zero or the INT instruction generate *exceptions*.

The 8088 reserves the lower 1024 bytes of memory for an interrupt vector table. There is one 4-byte vector for each of the 256 possible interrupt/exception numbers. When an interrupt or exception occurs, the processor: (1) clears the interrupt flag in the PSW, (2) pushes PSW, CS, and IP (in that order), (3) loads IP and CS (in that order) from the appropriate interrupt vector location, and (4) transfers control to that location.

For external interrupts (IRQ or NMI) the interrupt number is read from the data bus during an interrupt acknowledge bus cycle. For internal interrupts (e.g. INT instruction) the interrupt number is determined from the instruction.

The `INT` instruction allows a program to generate any of the 255 interrupts. This ”software interrupt” is typically used to access operating system services.

Exercise: MS-DOS programs use the INT 21H instruction to request operating system services. Where would the address of the entry point to these DOS services be found?

The `CLI` and `STI` instructions clear/set the interrupt-enable bit in the PSW to disable/enable external interrupts.

The `IRET` instruction pops the IP, CS and PSW values from the stack and thus returns control to the instruction following the one where interrupt or exception occurred.

Pseudo-Ops

A number of assembler directives (“pseudo-ops”) are also required to write assembly language programs. `ORG` specifies the location of code or data within the

segment, DB and DW assemble bytes and words of constant data respectively.

Segment/Offset Addressing

Since address registers are only 16 bits wide they are always used together with with one of the segment registers to form a 20-bit address. This is done by shifting the segment register value left by 4 bits before adding it to the address register. DS (data segment) is used for data transfer instructions, CS (code segment) is used with control transfer instructions, and SS is used with the stack pointer.

Exercise: If CX contains 1122H, SP contains 1234H, and SS contains 2000H, what memory values will change when the PUSH CX instruction is executed?

The use of segment registers reduces the size of pointers to 16 bits. This reduces the code size but also restricts the addressing range of a pointer to 64k bytes. Performing address arithmetic within data structures larger than 64k is awkward. This is the biggest drawback of the 8088 architecture.

We will restrict ourselves to short programs where all of the code, data and stack are placed into the same 64k segment (i.e. CS=DS=SS).

Byte Order

When multi-byte values are stored in memory the bytes are stored with the least-significant byte at the lowest memory location (so-called "little-endian" format). This is the opposite of the "big-endian" order used by Motorola processors.

Exercise: If memory locations 1000 and 1001 have values 01 and 02, what value will be loaded into AX if the 16-bit word at address 1000 is loaded into AX?

8088 Assembly Language

The operand format is *destination*,*source*, e.g. `MOV AX,13H` loads AX with 19.

The assembler keeps track of the type of each symbol and uses it select immediate or direct addressing. This can be changed by using the OFFSET operator to convert a memory reference to a 16-bit value. For example:

```
MOV  BX,COUNT        ; load value of COUNT
MOV  BX,OFFSET COUNT ; load location of COUNT
```

Hex and binary constants are indicated by using an H or B suffix after the number. A leading 0 must be added if the first digit of a hex constant is not a digit.

Example

This is a simple program that demonstrates the main features of the 8088 instruction set. It uses the INT operation to invoke MS-DOS to write characters to the screen.

```
; Sample 8088 assembly language program.  This program
; prints the printable characters in a null-terminated
; string (similar to the unix ("strings" program).

; There is only one "segment" called "code" and the
; linker can assume DS and CS will be set to the right
; values for "code". The code begins at offset 100h
; within the segment "code" (MS-DOS .COM files).

code segment public
     assume  cs:code,ds:code
     org     100h

start:
    mov     bx,offset msg   ; bx points to string
loop:
    mov     al,[bx]         ; load a character into al
    cmp     al,0            ; see if it's a zero
    jz      done            ; quit if so
    cmp     al,32           ; see if it's printable
    jl      noprt           ; don't print if not
    call    printc          ; otherwise print it
noprt:
    inc     bx              ; point to next character
    jmp     loop            ; and loop back

done:
    int     20h             ; return to DOS

; subroutine to print the byte in al

printc:
    push    ax              ; push ax and dx
    push    dx
    mov     dl,al           ; use DOS to
    mov     ah,02H          ; print character
    int     21H
    pop     dx              ; restore ax and dx
    pop     ax
    ret

msg  db     'This',9,31,32,'is',20H,'a string.',0

; example of how to reserve memory (not used above):

buf  db      128 dup (?)     ; 128 uninitialized bytes
```

```
code ends
     end     start
```