What´s inside an 8086

CPU - Central Processing Unit
BIU - Bus Interface Unit

Also: a little review for exam 2

What you should be able to do
Chapter 4

Understand and be able to explain (UCE)

- Any of the common number representations
- Each of the 4 functions, plus shifts, in the various representations
- Why unsigned multiplication is different than signed
- Carry propagation, why and how done
- Be able to program operations using the MIPS floating-point coprocessor

If you are clever

- Be able to describe how fixed to/from floating conversion works – this is the first part of problem 2
- How would carry-save multiplication work with negative operands
- Compare performance of the various approaches
- Understand how the MIPS instruction set is made expandable to coprocessors

What you should be able to do
Chapter 5

- Trace through the operation of any of the classes of instruction
- Explain the operation of the microcode
- Be able to analyze how the immediate/displacement/branch field should be handled – shifting, sign extension, etc.
- If you are clever
  - Be able to describe how to add a new variety of instruction or those not described (shifts, multiplication, division)
  - Describe how and why only some of the branches and sets are real rather than pseudoinstructions
  - Compare performance of the various approaches

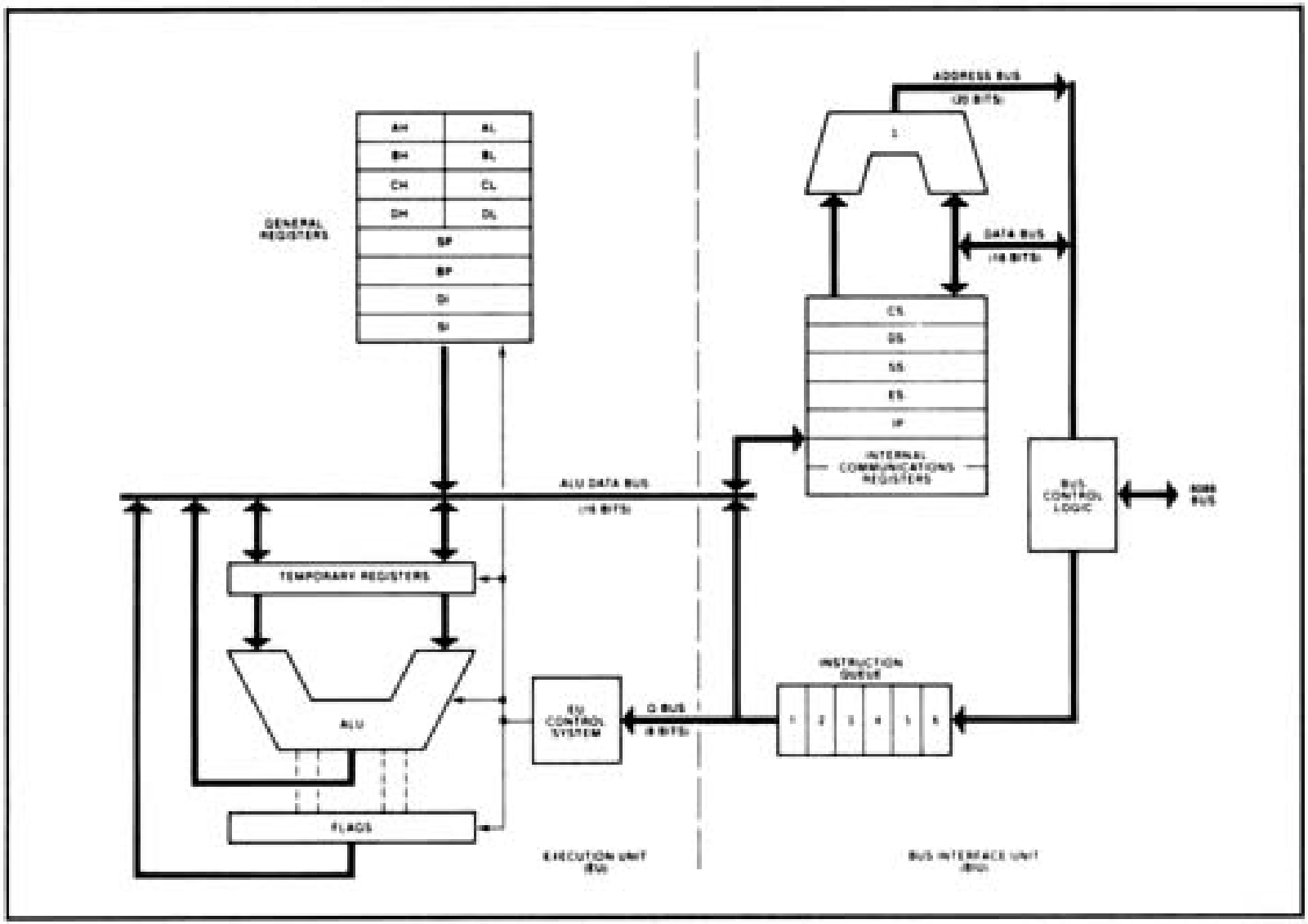


Figure 1-3 8086 Simplified Functional Block Diagram

The two parts of an 8086

CPU

- Interprets instructions
- Does arithmetic and logic
- Calculates effective address
- Specifies which memory segment to use

BIU

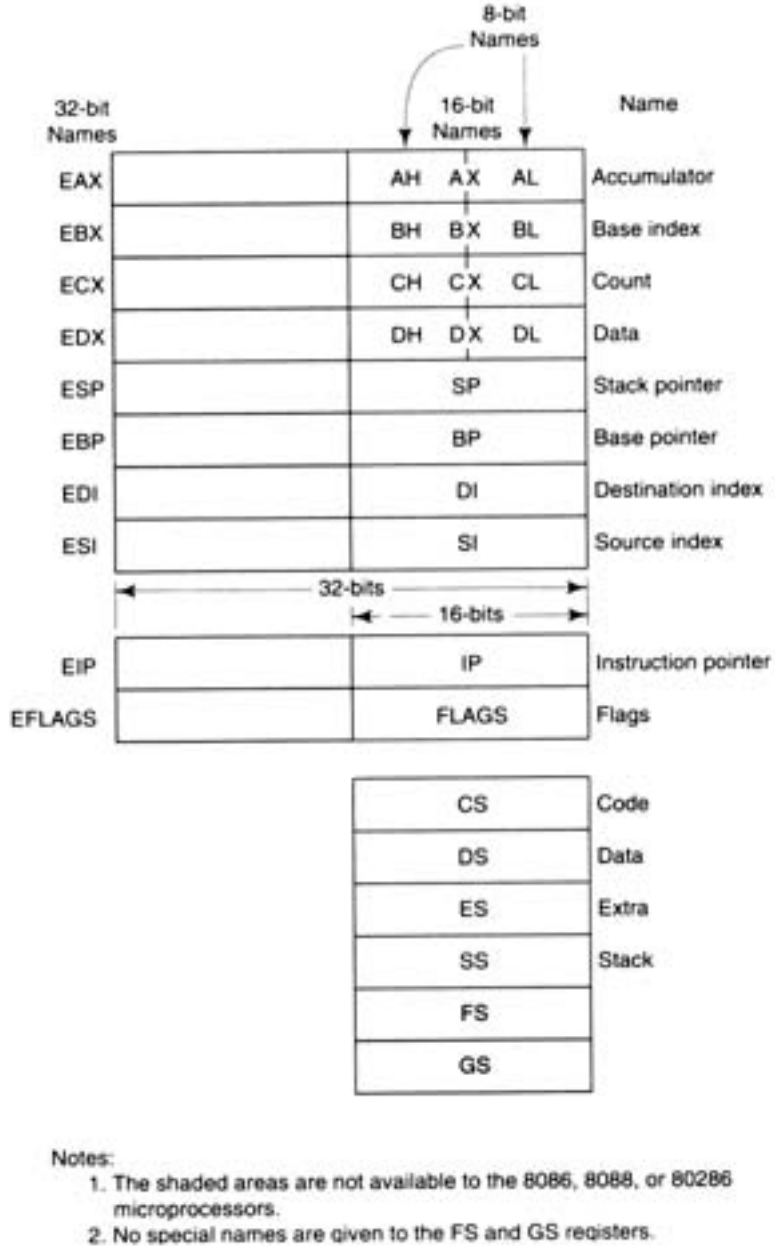
- Fills instruction pipeline - ahead of time
- Calculates physical address using effective address (EA) and segment register contents, that´s why it has its own adder
- Handles the external address and data buses

The CPU Registers

- This shows 80386 and up
- 8086/8 have only 8 and 16 bit registers and operations

The BIU has 16-bit segment registers

- code
- data
- stack
- extra



The CPU Registers

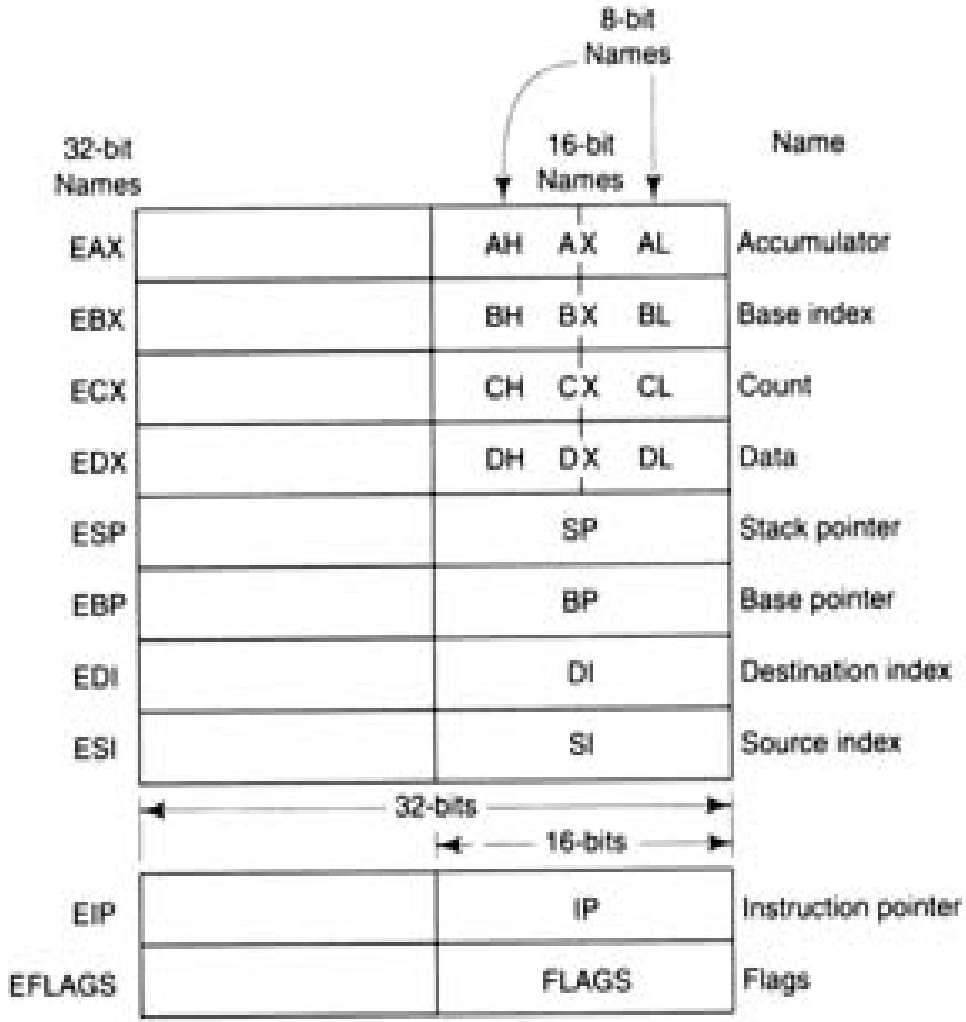
For the 8086 - these registers are 16 bits

Some have 8-bit halves

8-bit - AL, AH, etc

16-bit - AX, SI, etc

32-bit - EAX, ESP, etc.

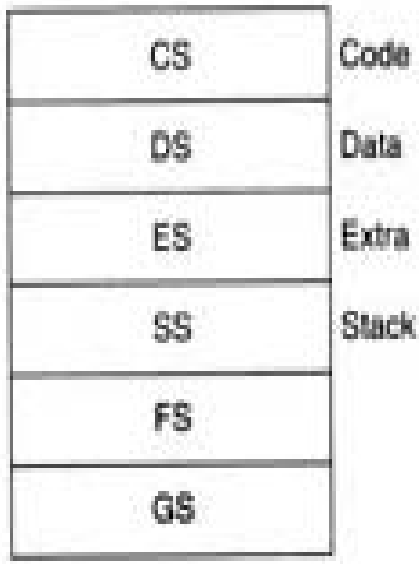


The BIU Registers

FS, GS are new with 80386

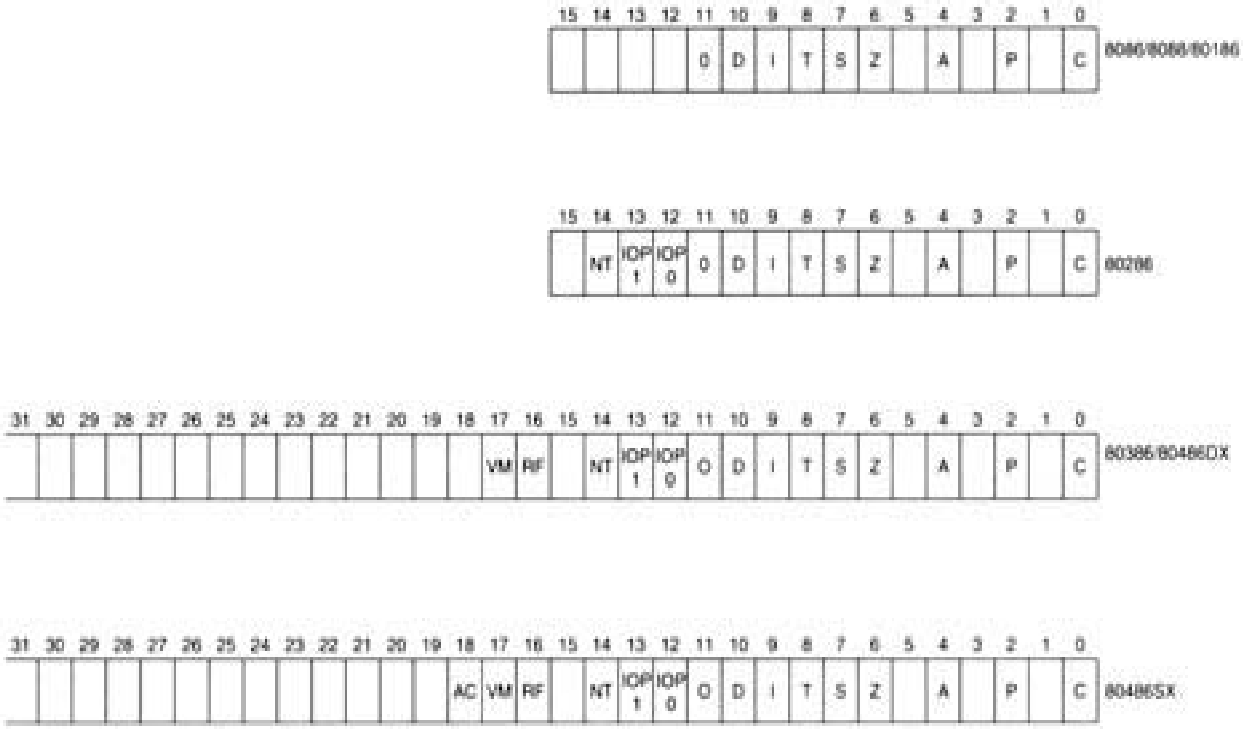
All are segment descriptors i protected mode (80286 and above)

All are segment bases in real (8086) mode



Notes:

1. The shaded areas are not available to the 8086, 8088, or 80286 microprocessors.
2. No special names are given to the FS and GS registers.



Note: The blank bits are reserved for future use and must not be defined.

FIGURE 1–8 The flag registers of all family members. Note how they are all upward compatible.

Status Registers - through 80486

Status registers remember results of previous operations

They govern whether conditional jumps are made

Evolution of the 8086 family

- 8086 - as shown, 8087 floating point unit
- 80186/8 - an integrated controller
  - 8086 instruction set
  - Built-in I-O ports and interrupt controller
- 80286, 80287
  - Protected mode
    - task switching
    - 16 MB address space

More evolution

- 80386
  - 32-bit registers and operations
  - 4 GB address space
  - 16 or 32-bit data bus
- 80486
  - Built-in cache memory
  - Built-in FPU

The Pentiums

CPU is a RISC machine, not an ‘86

Two pipelines, one can do floating operations

Hardware interprets 8086 code into RISC

Double-speed clock

Built-in debugging capability

Pentium Pro, II, and III have Multimedia instructions

```
DATA    SEGMENT PARA 'DATA'
        ORG     7000H
POINTS  DB      16 DUP(?)               ;save room for 16 data bytes
SUM     DB      ?                       ;save room for result
DATA    ENDS

CODE    SEGMENT PARA 'CODE'
        ASSUME CS:CODE, DS:DATA
        ORG     8000H
TOTAL:  MOV     AX,7000H                ;load address of data area
        MOV     DS,AX                           ;init data  segment
register
        MOV     AL,0                    ;clear result
        MOV     BL,16                   ;init loop counter
        LEA     SI,POINTS               ;init data pointer
ADDUP:  ADD     AL,[SI]                         ;add data  value to
result
        INC     SI                      ;increment data pointer
        DEC     BL                      ;decrement loop counter
              JNZ     ADDUP                   ;jump if counter not zero
        MOV     SUM,AL                  ;save sum
        RET                             ;and return
CODE    ENDS
        END     TOTAL
```

Memory Addressing in the 8086

How the 8086 (and higher 80x86 in real mode)
see the memory
Four movable windows, each of 64K
Code, Data, Stack, and Extra

Programmer's view of the world

CPU Registers
AX=AH,AL
BX=BH,BL
CX=CH,CL
DX=DH,DL
BP
SP
SI
DI

I/O space
64K ports
Reached only by
IN and OUT
instructions

BIU Registers
CS
DS
SS
ES
Instruction pipeline

Control Unit
in CPU
Contains IP, flags
receives instructions
Controls everything

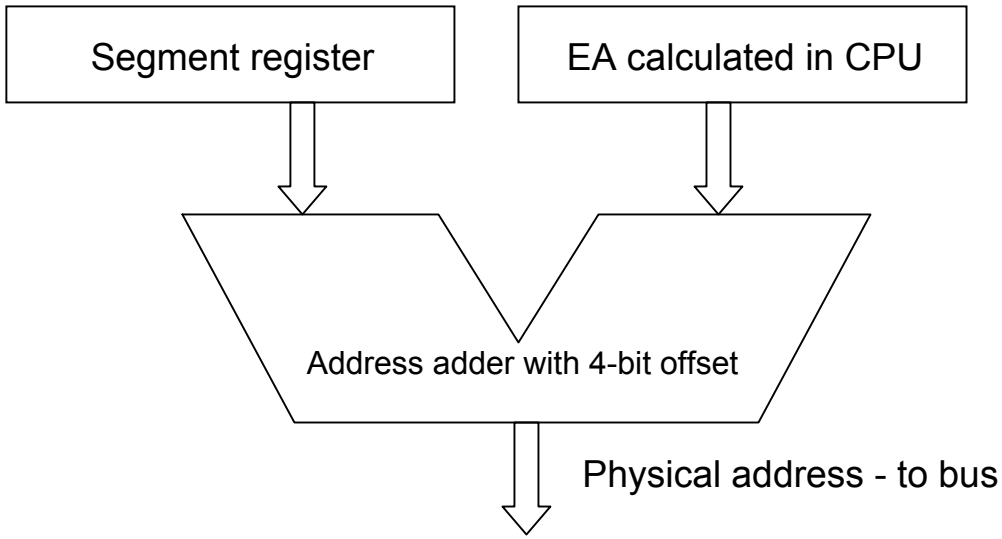
Code Segment
can also contain data
Pointed to by 16*CS

Data Segment
default for BX, SI, DI
pointed to by 16*DS

Stack Segment
Default for BP based
Used by stack instructions
Pointed to by 16*SS and SP

Extra Segment
Destination for string instructions
Default for nothing else
Pointed to by 16*ES

Memory operands



Memory address has up to three parts

Displacement is part of the instruction

Base - if specified - is contents of BX or BP

Index - if specified - is contents of SI or DI

Effective address calculations are 16-bit - they wrap around

Some typical formats
A[BX] - based with displacement
WORD PTR [BP] - based only, in stack segment
[BP+4][SI] - based and indexed with displacement
[BX+4+DI] - based and indexed with displacement - in data segment
A[10] - displacement only - in data segment

Memory Address Calculation Examples

DS	2314H
SS	9000H
BX	3255H
BP	0FFF2H
DI	0EEEEH

DS:[BP+3+SI]

23140 - DS
0FFF2 - BP
3
0EEEE - DI
0EEE3 - EA=[BX+3+DI]
32023 = EA+16*DS

[BP-4]

90000H - 16*SS
0FFF2H - BP
0FFEEH - EA=BP-4
9FFEEH - EA+16*SS

[BX+20H]

23140H - 16*DS
03255H - BX
00020H - displacement
03275H - EA=BX+20H
263B5H - EA+16*DS

A Few Coding Examples

```
; This is the basic program template in the new assembler format
; It can be used as a pattern for most of your programs
            .model      small                   ; the small model has one code and one data
            .stack      100H                    ; segment and results in a .EXE after linking
            .data
; data definitions go here
            .code
begin:      mov         AX, DGROUP              ; DGROUP is a proper name in the small model
            mov         DS,AX                   ; this sets DS so the data segment can be accessed
; all the rest of your code goes here
            .startup
            .exit
            end         begin
```

```
 ; For most examples from now on the data and code will be shown without the segment setup
; An example that right shifts an array one place
            .data
NWORDS  EQU     30                  ; Locality of reference - define it once
ARRAY   DW      NWORDS DUP (?)      ; An array of NWORDS words
            .code                   ; Leaving out the segment setup
            MOV     BX,NWORDS       ; This is immediate-to-register
            MOV     CX,BX           ; Register-to-register is faster
            SHL     BX,1            ; Multiplying BX by 2, we are using words
            CLC                     ; shifting in an 0 initially
startshift: DEC     BX              ; Using DEC twice lets the carry alone
            DEC     BX
            RCR     ARRAY[BX],1     ; The carry is retained from shift to shift
            LOOP    startshift      ; LOOP uses CX and doesn't affect flags
```

Pointer operations

Pointers point to things

- Manufactured by
  - LEA, PEA
- Used by indexed or based addressing
  - [BX], [BP], [SI], [DI]
- Example:

```
LEA BX, Array                ; BX  points to first element of array
ADD AX, WORD PTR [BX]  ; this adds current element to A
INC BX                                ; these step BX by one word
INC BX                                ; this is faster than adding 2 to BX
```

- This avoids using displacements
- More importantly, passing a pointer into a function is the way to make arrays and structures available inside the function

Processor Structure and Data Types

The data types and register set of the 80x86 family

These types are basically like those of most processors

CPU Basic Structure

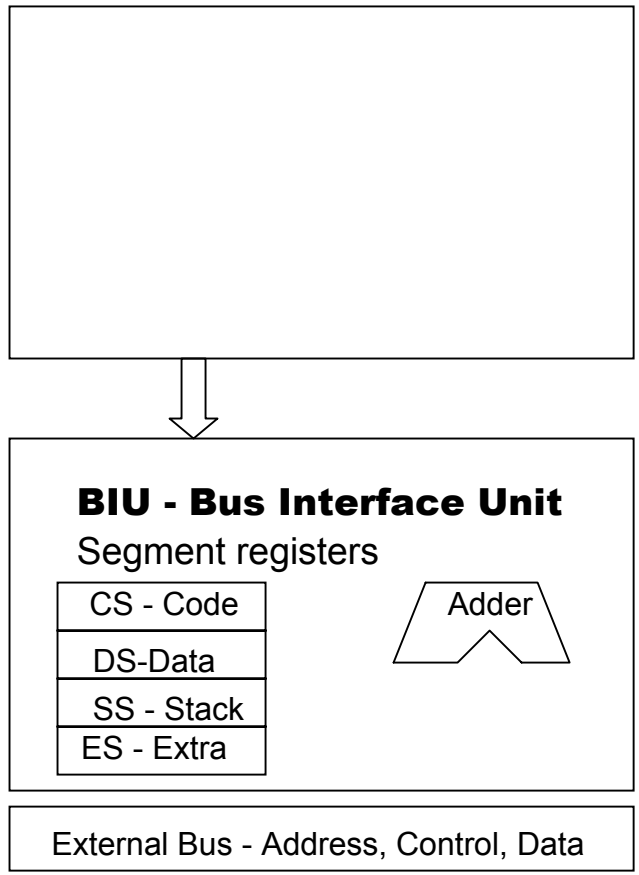
CPU - Central Processing Unit

- Gets instructions from pipeline in BIU
- Gets and puts data to BIU
- Sends logical address (which segment and effective address) to BIU
- Executes instructions

BIU - Bus Interface Unit

- Contains segment registers
- Contains instruction pipeline for prefetched instructions
- Calculates physical addresses from effective addresses and segment

CPU and BIU basics



The Processor's Data Types

- **Integer**
  - Two's complement notation
  - 8 (B-byte), 16 (W-word), 32 (D,double)
- **Unsigned**
  - Interpreted as positive or zero
  - 8 (B-byte), 16 (W-word), 32 (D,double)
- **Floating**
  - Sign, exponent, magnitude
  - 32 (D,double), 64 (Q-quad, long IEEE notation), 80 (T-ten byte)
- **Pointers point to something**
  - Near, same segment
  - Far, pointer contains offset and segment value
- **Data is what the instruction interprets it as being**
  - It is just a bit pattern, that could be anything the current instruction wants it to be

Operations on Data Types

- **Integer**
  - Add/subtract and logic - two operands
  - Multiply/divide - one operand in AX,DX, other is an operand
- **Unsigned**
  - Add/subtract are same operation as integer
  - Multiply/divide are different operations, same source and destination
- **Decimal operations can be done on unsigned bytes**
  - BCD format is two digits/byte, add,sub only
  - ASCII (misnamed) format is one digit per byte, four functions
- **Floating point**
  - Done in the coprocessor, which has its own registers and instructions
  - Not covered in this course
  - Coprocessor also does big decimals, trig, and big integers

Instruction Groups - in the order we will teach them

- Move instructions (Not including string)
  - Operands include
    - register
    - memory
    - immediate
- Two-operand arithmetic and logical
- One and zero-operand arithmetic and logical
- Conditional jump instructions
- Stack operations
- Multiply and divide
- The decimal feature
- Jump, call and return
- The string feature

Move instructions

Moves byte, word, or double data from one place to another

No effect on flags

See picture for possible data paths

Assembler identifies type and size of move from operands

Only moves and stack instructions can reach segment registers

Only one operand can be memory or immediate

Add/Subtract and Logical

All are two-operand instructions

All affect flag registers

Data paths are like move, except status registers are not affected

Effect is like C language

    dest op= source

Add/subtract affect all flags

Logicals do not modify carry or overflow

Add/subtracts are

    ADD, ADC, SUB, SBB, CMP

Logicals are

    AND, OR, XOR, TEST

One operand Arithmetic and Logical

- Arithmetic one-operand
  - NEG, INC, DEC
- Logical one-operand
  - NOT
- Arithmetic zero-operand
  - CBW, CWD, DAA, DAS, AAA, AAS, AAM, AAD
- Multiply and divide
  - Word and byte size
  - Byte size uses AH, AL, and operand
  - Word size uses DX, AX, and operand
  - AAM follows multiply
  - AAD precedes division

Conditional jumps

- All jump based on the state of the flags
  - Example JNE jumps if last operation didn't result in zero
- It is easier to remember the adjectives than the logic
  - For signed operations, L (less), E (equal), G (greater)
  - For unsigned operations B (below), E (equal) or A (above)
  - C is carry or borrow, and also represents unsigned overflow
  - O is overflow for signed operations, meaningless for unsigned
- Conditional jumps can jump only 128 bytes forward or back
  - Assembler tells you if you tried to jump too far
  - Fix this by using a long jump and changing the sense of the conditional
- Logical and loop instructions generally leave carry alone
  - This feature is needed for fast multiword arithmetic and shift operations