Session 03

8085 Instruction Set

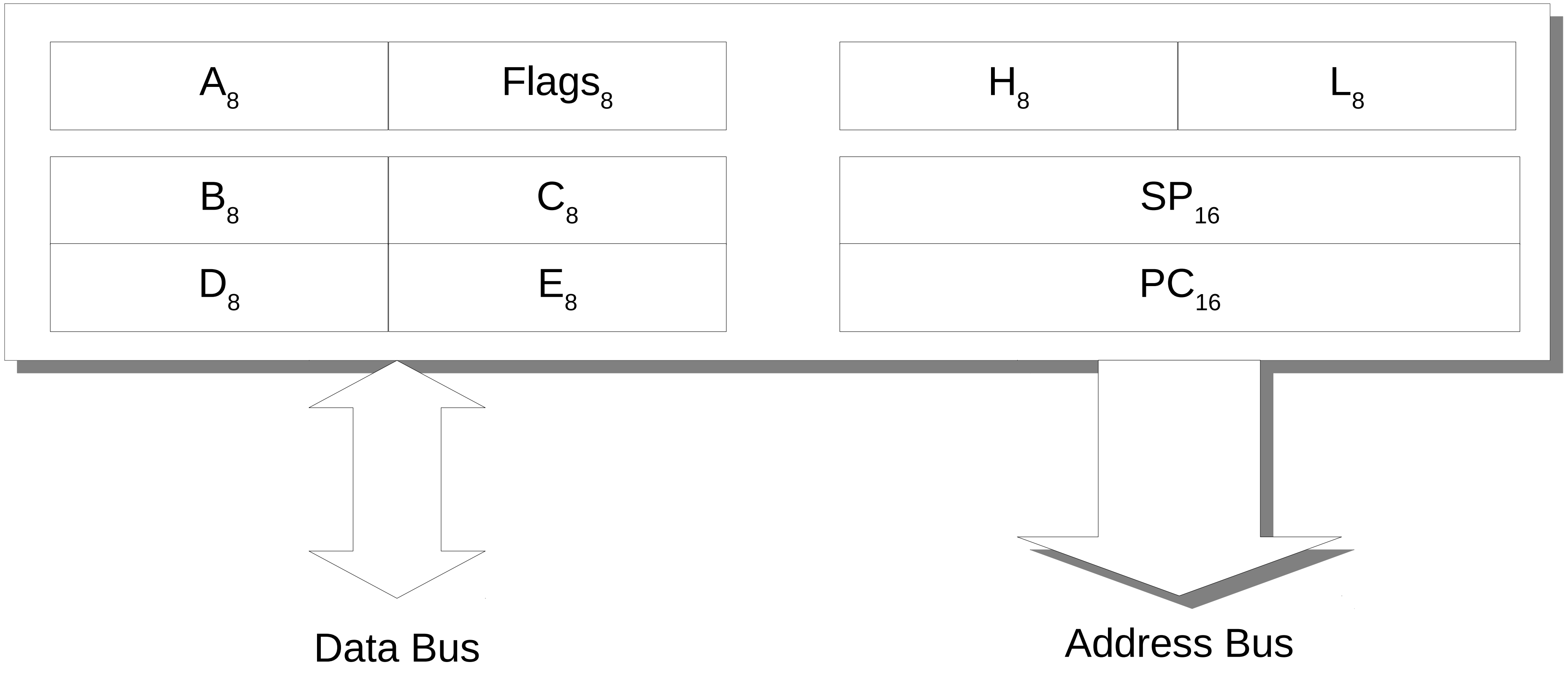
Overview

- Recap of 8085 Architecture
- 8085 Programming Model
- Encoding of Instruction
- 8085 opcodes and arguments
- 8085 Opcode codes
- Assembly Code

Recap

- Evolution of Microprocessors – VLSI, densification, Intel 4004, Intel x86 series
- Architecture of 4004 and 8085 processors
  - ALU, Accumulator, Flags, Registers, Bus Control (Address/Data)
  - Timing and Clock, Interrupts
  - External Bus Interfaces – Memory and Peripherals (“Southbridge”)
  - Peripheral Controllers – DMA, Timers, Interrupts, Serial Comm., ...
- Addresses and Data – Hexadecimal encoding
- ESA Kit – different parts, operations
- Simple 8085 program – multiplication through repeated addition using gnusim8085

8085 Programming Model



8085 - C structure

```
struct i8085 {
        union {
                struct {
                        unsigned char c, b, e, d, l, h;
                        unsigned char flag, a;
                } byte;

                struct {
                        unsigned short bx, dx, hl;
                        unsigned psw;
                } word;

                unsigned char r8[8];

                unsigned char r16[4];
        } regs;
        unsigned short sp;
        unsigned short pc;
        unsigned short temp;
        unsigned char imask;
} __attribute__((packed)) esa;
```

```
esa.regs.byte.a += esa.regs.byte.b

esa.regs.word.hl

esa.regs.r8[2]

esa.regs.r16[1]

esa.sp = esa.regs.word.hl

esa.pc = esa.regs.word.hl

esa.temp = esa.regs.word.de;
esa.regs.word.de = esa.regs.word.hl
esa.regs.word.hl = esa.temp
```

8085 Instructions

- 8-bit, bit 0 is least significant
- opcode is always 8-bit
- instruction length varies
  - op
  - op, data
  - op, low, high
- 16-bit is always low-endian
- no alignment restrictions
- Memory address space – 16-bit
  - 0000H..FFFFH
- Input/Output Port space – 8-bit
  - 00H..FFH
  - duplicated onto address bus
- Registers B, D, H may be treated as 8-bit or extended into 16-bits
- PSW = A+Flags

4200H	op
4201H	lo8
4202H	hi8
4203H	..

8085 opcode, args

- opcode – variable bits
- Registers – 3bits
  - B= 0, C=1, D=2, E=3
  - H=4, L=5
  - [HL]=6
  - A=7
- Extended Registers - 3bits
  - BC=0, DE=2, HL=4, SP=6
  - [HL]=6
- Condition Flags – 3bits
  - NZ=0, Z=1
  - NC=2, NC=3
  - PO=4, PE=5
  - P=6, M=7

1	0	0	0	0	r	r	r

ADD r – add register to accumulator
83H – ADD E
86H – ADD [HL], ADD M

1	1	c	c	c	0	1	0

Jxx – Jump Conditional
C2H JNZ i16
CAH JZ i16
...

Note: Initially, opcodes were designed around octal (3-bit) encoding, not hexadecimal. 8085 opcodes were influenced by PDP11 ISA.

Opcode types

- Data transfer, Arithmetic, Logic, Branch and Control
- Data Transfer – copy 8-bit or 16-bit between register<->register, register<->memory
    - They don't affect flags - increment/decrement/complement, bit rotate?
- Arithmetic operations – add, subract, rotate (no multiply or divide)
- Logical Operations – bits not bytes
  - and, or, xor, compare. They affect flags
- Branching (changes PC) – jump, call, return
- Machine Control – stack management, interrupts (enable, disable, mask), input, output, halt
- Ref: 8085-hexcodes.pdf in Notes folder in LMS

Assembly Code

- Microprocessors and microcontrollers used different encodings for instructions (even from the same manufacturer)
- Programming in machine code is tedious and error-prone.
- So mnemonics were used to build a program in a text file and a tool called *assembler* was used to translate text in this file into machine language instructions. The textual language came to be known as **Assembly Language***.*
- But then, different vendors started using different mnemonic codes for same instructions, so assembly language was not *portable* across assemblers .
- This forced microprocessor vendors to standardize on instructions and provide *backward compatability*.
- Intel introduced a stable instruction set with 8086 and licensed it to other vendors. This became known as the Intel x86 Instruction Set Architecture.

Lab Exercises

- Watch out for the following during your lab exercises
- 8085 opcodes are different from 8051 opcodes. Make sure you are using the right kit
- Learn to convert hex to binary and binary to hex (4-bits and 8-bits).
- Learn to map key decimal sizes to Hex. It is easy to miss a 0 and make a mess.
  - 0x100 is 256, 0x1000 is 4K, 0x1000 is 16K, 0x8000 is 32K
  - 1024 is 0x400
- Learn to distinguish different encodings – 8085 opcodes, ASCII, 7-segment LED code
  - Everything is just a number to the processor. Meaning is upto us.
- Reading the contents of HL is different from reading the byte in the address contained in HL (aka [HL])
- Not all operations affect the Flags. Read up on it.
- When making a call, the caller has to preserve the registers it uses.

Session 04

8085 Opcodes

Overview

- Overview of Instruction Sets
- Data Transfer Group
- Arithmetic Group
- Logical Group
- Branch Group
- Control Group

Data Transfers

- MOV r1, r2 – copy data from register r2 to r1
- MVI r, i8 - copy next byte to register r
- LDA a16 – copy byte from an address to accumulator
- STA a16 – copy accumulator to memory at address
- LHLD a16 – copy 16-bits from address to HL registers
- SHLD a16 – copy HL registers to memory at address
- LXI r, i16 – copy next 16-bits to a register pair
- LDAX rr – copy data from address in register pair to accumulator
- STAX rr – copy accumulator to memory address in register pair
- Stack and HL register ops (see later)

Arithmetic Group

- ADD r – add register to accumulator
- ADI i8 – add next byte to accumulator
- ADC r – add register to accumulator along with carry bit
- ACI i8 – add next byte to accumulator along with carry bit
- SUB, SBI, SBB, SBI - likewise
- INR r, DCR r – increment/decrement register contents by one
- INX r, DCX r – increment/decrement register pair contents by one.
- DAD rr – add register pair to HL register pair
- DAA – decimal adjust accumulator. used along with AC flag for Binary-Coded Decimal arithmetic.

INR/DCR does not affect Carry flag (hint: see bit rotate ops)
INX/DCX does not affect any flag.
DAD affects only Carry flag

Logic Group

- Boolean bit-wise operations
- ANA r, ORA r, XRA r – logical AND, OR, XOR register with accumulator
- ANI i8, ORI i8, XRA i8 – logical AND, OR, XOR register with next byte
- CMP r – compare register bits with accumulator.
- CPI i8 – compare next byte with accumulator.
- These ops affect all flags
- RLC, RRC – rotate accumulator bits left, right (aka logical rotate)
- RAL, RAR – rotate accumulator and carry bit (9th bit) (aka arithmetic rotate)
- STC/CMC – set or complement carry flag (what about clear?)
- These flags affect only carry flag
- CMA – complement bits in accumulator. does not affect flags

Branch Group

- Affects PC register
- JMP i16 – copy next 16 bits to PC
- CALL i16 – push PC on stack and copy next 16-bits to PC
- RET - pop top of stack into PC
- Conditional flags – Zero, NonZero, Carry, NoCarry, ParityOdd, Parity-Even, Plus, Minus
  - eg. JZ, JNZ, JC, JNC, JPO, JPE, JP, JM
- PCHL – copy HL to PC
- RST n – push PC onto stack and set PC to 8*n (aka Soft Interrupt)
- Special 16-bit operations
  - SPHL – copy HL to SP
  - PUSH rr, POP rr – push register pair onto stack, pop top of stack onto register pair
  - XTHL – exchange top of stack with HL
  - XCHG – exchange DE with HL

Control Group

- Controls operation of all other subsystems, buses and pins
- OUT a8, IN a8 – accumutor <-> I/O port. 8-bit address is duplicated on Address bus.
- DI, EI – disable or enable all Interrupts. Interrupts are disabled on RESET
- HLT – halt, all bus lines are tri-stated (i.e. disconnected)
- SIM/RIM – set or reset interrupt masks

Note: not all 8-bit codes are valid opcodes. Effects of executing an invalid opcode is undefined.

All together

- Programs are created by stringing together a sequence of instructions, initializing PC register to the start of the code and then letting it rip. It is upto to the program to yield control back to the BootROM.
- Four basic types of sequences
  - simple sequence, S = *i1, i2, i3, i4*. ... which are executed one after another
  - conditional sequence: *if C then S1 else S2*
  - Loop sequence: *while C then S1 else S2* or *do S1 until C*; .....
  - Subroutine calls: *call F1*
- An interrupt can break this sequence and transfer control to a handler.
- Typical errors:
  - Off by one – count one more or one less than what is needed
  - Incorrect or Incomplete logical test
  - Overrun or Infinite loop – where the condition may never be satisfied.
  - Challlenge – coming up with the right *Invariant*