Tutorial: Instruction Set Architecture Continued

(ISA)

Yul Williams, D.Sc.

Goals

- At the conclusion of this session, the student should be able to:
  - Understand the design of an ISA
  - Understand the structure of the Intel 8085 ISA
  - Understand the concepts of addressing modes
  - Understand the basic classes of ISA instructions

Outline

- ISA Design Issues
- The Relatively Simple ISA
- Instruction Formats
- The Intel 8085 Microprocessor
- A Detailed Look at the 8085 ISA

ISA Design Issues

ISA Design Considerations

- One of the most important aspects in the design of a microprocessor is the design of its ISA
- Think ahead
  - What kinds of tasks is the processor expected to handle?
  - What kinds of tasks would you like it to do?
  - Is this a general purpose or application-specific microprocessor design?

General Purpose Microprocessor ISA

- A general purpose microprocessor's ISA is …well…general!!
  - It will most probably contain a large number of instructions to handle a wide variety of programming tasks
  - It may also have a large number of registers to accommodate many of the programming tasks

Application Specific Microprocessor ISA

- Some microprocessors may be designed for very simple and/or specific purposes.
- The purposes may range from microwave oven control to navigational control for commercial airliners
- The ISA for these application specific microprocessors may be considerably less robust than its general purpose counterpart
  - Smaller number of instructions
  - Reduced number of registers
  - Smaller address space

Completeness

- Completeness is a term used to describe an attribute of a microprocessor's ISA
- Does it have the sufficient number and types of instructions to perform the desired tasks?

Orthogonality

- Some microprocessor ISA's provide a large set of instructions
- There may exist small subsets within the instruction set that perform the same functions. In short, the instructions overlap each other's functionality
- Instructions are orthogonal if the do not overlap
- A good instruction set minimizes the overlaps

Effects of Orthogonality on Hardware

- Some instruction sets include a large number of Jump instructions
  - Jmp; unconditional jump
  - Jnz; conditional –if previous result was not zero
  - Jz; conditional –if previous result was zero
  - Jp; condition –if previous result was positive
- Each instruction results in more hardware elements in the microprocessor to handle each additional Jump instruction
- Increased orthogonality means reduced hardware design

Registers and Hardware Complexity

- Every microprocessor has a basic set of registers
- The number of registers varies with the intended purposes of the microprocessor
- Each register requires additional circuitry to handle loading and unloading data into and out of the register spaces
- If you are designing a microprocessor that counts the number of cars passing through a highway toll gate, a couple of registers may be all that is required.
- More complex microprocessors, however, may have literally hundreds of registers (e.g. Intel Itanium processor)

Floating Point Capabilities

- General purpose microprocessors almost always require the capability to perform floating point calculations
- This requires logic to perform the operations and floating point data registers to hold the floating point data types
- These features significantly increase the complexity of the instruction set design
- The instructions to perform floating point operations must be augmented to the instruction set
- Floating point operations may not be a factor for consideration in some application specific microprocessors

Backward Compatibility

- One of the most important factor for widely used and general purpose microprocessors is the issue of backward compatibility
- Each new processor (in a series or family) must be able to leverage existing programs
- In order to facilitate this requirement, the new microprocessor must include the instructions from the previous version(s) of the microprocessor to ensure binary code compatibility
- This adds a tremendous design burden!

Data Sizes

- Based on programming background using High Level Languages (HLLs), we know that character, integer, floating point, and boolean data types have different storage requirements
- In addition to the storage requirements, they must be transported across the microprocessor busses and temporarily stored in internal registers
- The ISA must be planned with all of these considerations in mind

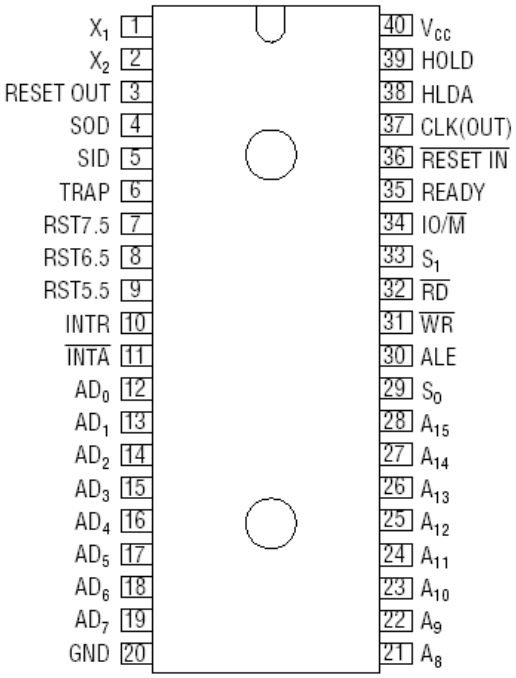
Miscellaneous Considerations

- Interrupts
  - If the microprocessor is expected to handle interrupts, the ISA must include instructions to handle this type of processing
- Conditional Instructions
  - Many processors include conditional instructions to handle the branching decisions
  - These branching decisions are often made with the use of flags. Flags are set by various operations and are examined by the conditional instructions in order to make branching decisions.

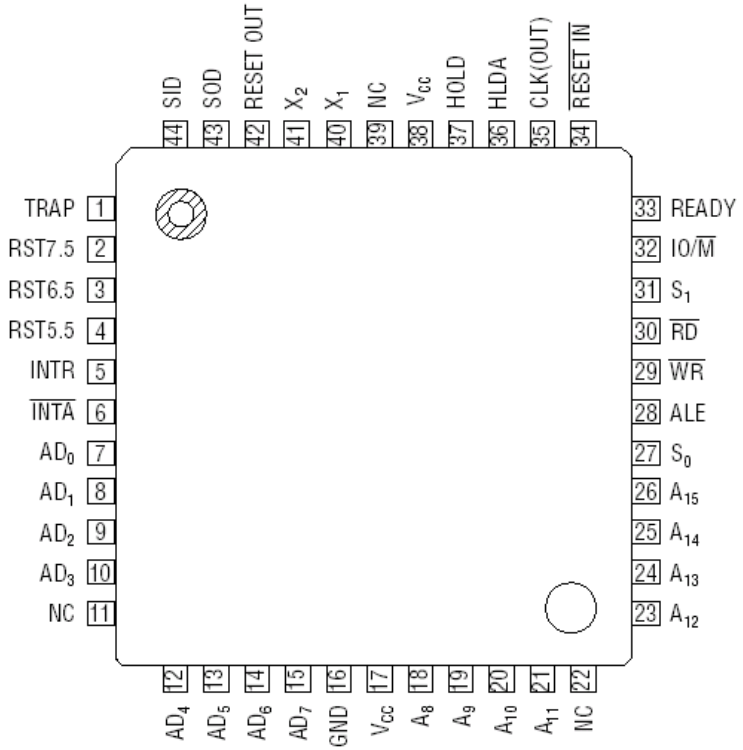
The Intel 8085 Microprocessor

8085 Packaging Examples

40 pin Plastic DIP



44 pin Plastic QFP



The Intel 8085 Microprocessor

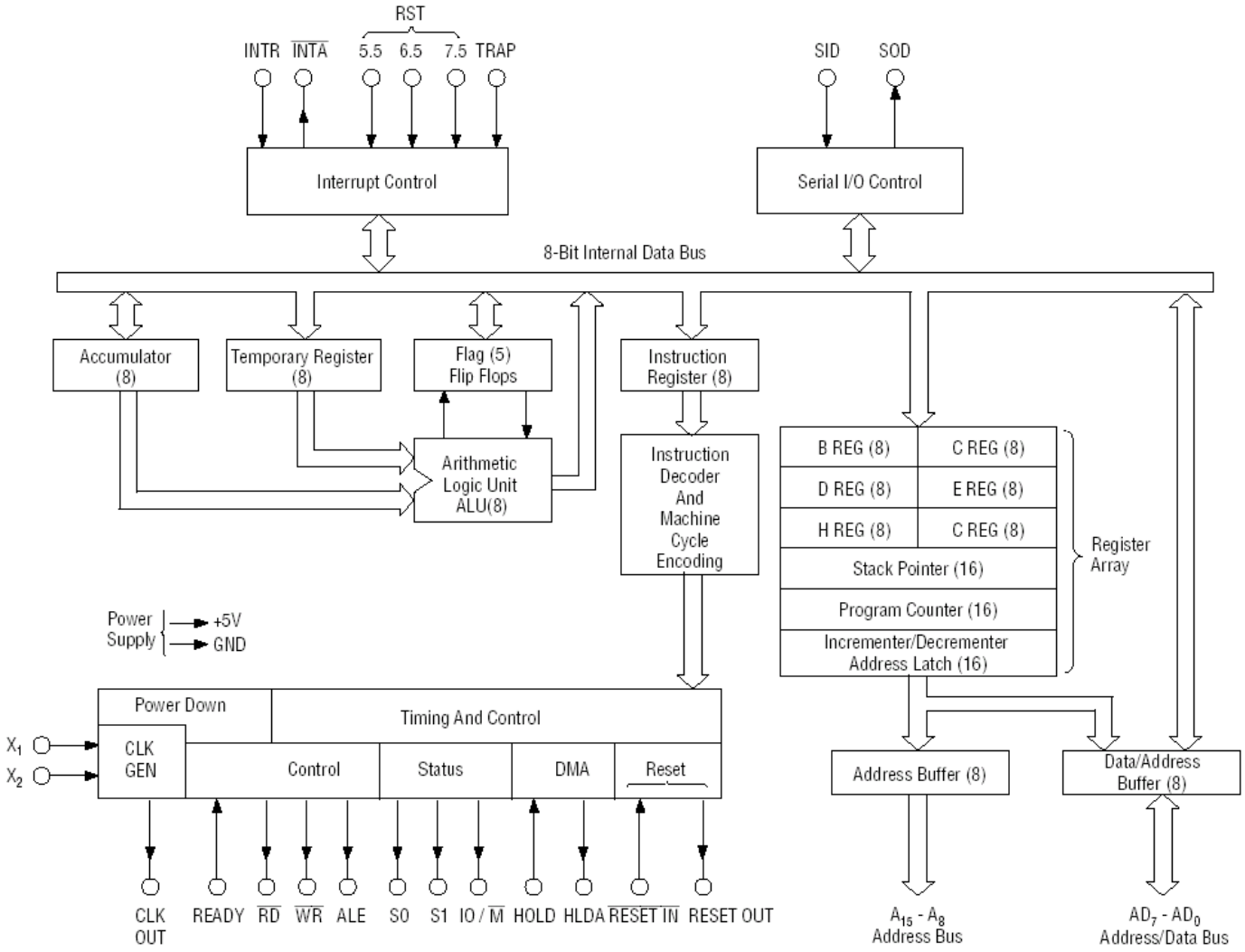
- The 8085 is the 3rd processor created and manufactured by the Intel Corporation in 1976
- It was preceded by the 8080 and the 4004 (the 1st recognized microprocessor)
- It was code compatible with its immediate predecessor and had a cadre of 74 instructions
- The top speed of the 8085 was 6.25 Mhz
- It was able to address up to 64KB of memory

The 8085

- Intel no longer produces the 8085 microprocessor, but they are widely available today on the commercial market
- Several other manufacturers have taken over the production of the 8085
- It is used widely by universities as a teaching model and is also used by hobbyists for small scale and low cost projects
- You can acquire an 8085 microprocessor on the market for less than $3.00 apiece

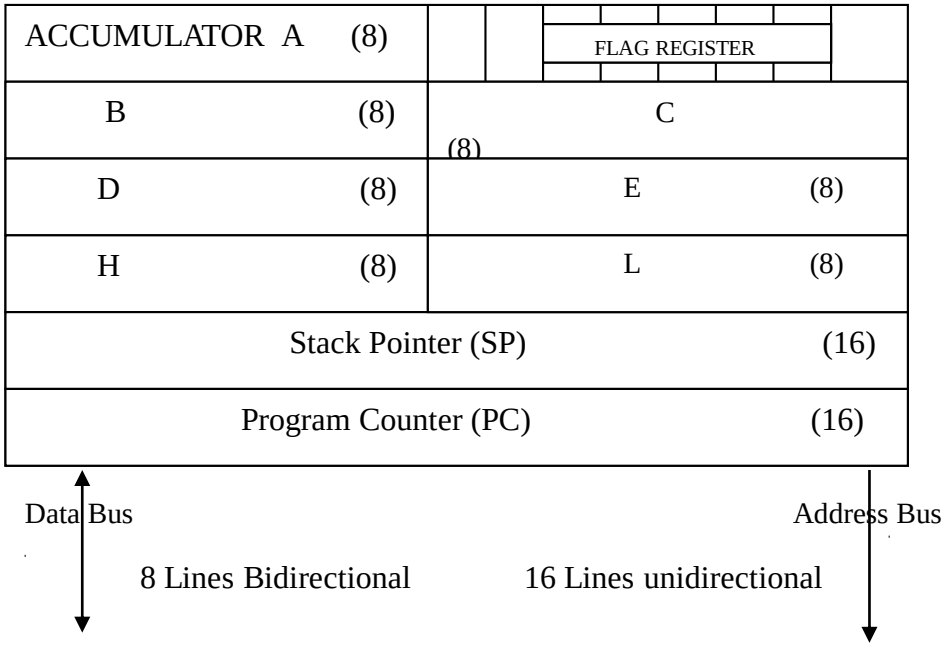
A Detailed Look at the 8085 ISA

Intel 8085 Microprocessor



Courtesy: OKI Semiconductors

The Registers



The 8085 has six general-purpose (and programmer accessible) registers to store 8-bit data; these are identified as B,C,D,E,H, and L as shown in the figure. They can be combined into register pairs - BC, DE, and HL - to facilitate 16-bit operations.

The Accumulator

The accumulator is an 8-bit register. This register stores 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also known as register A.

Flags

- The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero(Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags; their bit positions in the flag register are shown in the Figure below. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.
- For example,
  - after an addition of two numbers, if the sum in the accumulator is larger than eight bits, the flip-flop uses to indicate a carry -- called the Carry flag (CY) -- is set to one.
  - When an arithmetic operation results in zero, the flip-flop called the Zero(Z) flag is set to one.
- The flags are stored in the 8-bit register so that the programmer can examine these flags by accessing the register through an instruction.

The Program Counter (PC)

- This 16-bit register deals with sequencing the execution of instructions.
  - This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register.
- The function of the program counter is to point to the memory address from which the next byte is to be fetched.
  - When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location

The Stack Pointer (SP)

- The stack pointer is also a 16-bit register used as a memory pointer.
- It points to a memory location in R/W memory, called the stack.
- The beginning of the stack is defined by the loading of a 16-bit address into the stack pointer

8085 Addressing Modes

- **Immediate addressing**
  - Data is present in the instruction. Load the immediate data to the destination provided.
  - Example: MVI R,data
- **Register addressing**
  - Data is provided through the registers.
  - Example: MOV Rd, Rs
- **Direct addressing**
  - Used to accept data from outside devices to store in the accumulator or send the data stored in the accumulator to the outside device. Accept the data from the port 00H and store them into the accumulator or Send the data from the accumulator to the port 01H. This is a reference to I/O-mapped instructions. This method will be further explained when we reached the topic of I/O architectures.
  - Example: IN 00H or OUT 01H
- **Indirect Addressing**
  - This means that the Effective Address is calculated by the processor. And the contents of the address (and the one following) is used to form a second address.
  - The second address is where the data is stored. Note that this requires several memory accesses; two accesses to retrieve the 16-bit address and a further access (or accesses) to retrieve the data which is to be loaded into the register.

8085 Data Transfer Instructions

Instruction	Operation
NOP	No operation
MOV *r1*, *r2*	*r1* = *r2*
MOV *r*, *M*	*r* = M[*HL*]
MOV *M*, *r*	M[*HL*] = *r*
MVI *r*, *n*	*r* = *n*
MVI *M*, *n*	M[*HL*] = *n*
LXI *rp*, Γ	*rp* = Γ
LDA Γ	*A* = M[Γ]
STA Γ	M[Γ] = *A*
LHLD Γ	*HL* = M[Γ], M[Γ+1]
SHDL Γ	M[Γ], M[Γ+1] = *HL*

Instruction	Operation
LDAX *rp*	*A* = M[*rp*] (*rp* = *BC*, *DE*)
STAX *rp*	M[*rp*] = *A* (*rp* = *BC*, *DE*)
XCHG	*DE* ↔ *HL*
PUSH *rp*	Stack = *rp* (*rp* ≠ *SP*)
PUSH PSW	Stack = *A*, flag register
POP *rp*	*rp* = Stack (*rp* ≠ *SP*)
POP PSW	*A*, flag register = Stack
XTHL	*HL* ↔ Stack
SPHL	*SP* = *HL*
IN *n*	*A* = input port *n*
OUT *n*	output port *n* = *A*

8085 Data Operations Instructions

Instruction	Operation
NOP	No operation
MOV *r1*, *r2*	*r1* = *r2*
MOV *r*, *M*	*r* = M[*HL*]
MOV *M*, *r*	M[*HL*] = *r*
MVI *r*, *n*	*r* = *n*
MVI *M*, *n*	M[*HL*] = *n*
LXI *rp*, Γ	*rp* = Γ
LDA Γ	*A* = M[Γ]
STA Γ	M[Γ] = *A*
LHLD Γ	*HL* = M[Γ], M[Γ+1]
SHDL Γ	M[Γ], M[Γ+1] = *HL*

Instruction	Operation
LDAX *rp*	*A* = M[*rp*] (*rp* = *BC*, *DE*)
STAX *rp*	M[*rp*] = *A* (*rp* = *BC*, *DE*)
XCHG	*DE* ↔ *HL*
PUSH *rp*	Stack = *rp* (*rp* ≠ *SP*)
PUSH PSW	Stack = *A*, flag register
POP *rp*	*rp* = Stack (*rp* ≠ *SP*)
POP PSW	*A*, flag register = Stack
XTHL	*HL* ↔ Stack
SPHL	*SP* = *HL*
IN *n*	*A* = input port *n*
OUT *n*	output port *n* = *A*

8085 Program Control Instructions

Instruction	Operation
JUMP Γ	GOTO Γ
J*cond* Γ	If condition is true then GOTO Γ
PCHL	GOTO address in *HL*
CALL Γ	Call subroutine at Γ
C*cond* Γ	If condition is true then call subroutine at Γ
RET	Return from subroutine
R*cond*	If condition is true then return from subroutine
RSTn	Call subroutine at 8*n (n = 5.5, 6.5, 7.5)
RIM	*A = IM*
SIM	*IM = A*
DI	Disable interrupts
EI	Enable interrupts
HLT	Halt the CPU

Summary

- Specifying the ISA is one of the most important aspects of microprocessor design
- Completeness is essential in the specification of an ISA
- Instructions should be orthogonal to help reduce the amount of digital logic in the processor
- The processor must be designed with adequate numbers and appropriate types of registers