

ENS2257 Microprocessor Systems

Lecture 4

The 68HC11 and 8086 Microprocessors

Microcontrollers and Microcomputers

• Introduction

- Generally **microprocessors** are evolving in two directions, *performance* and *integration*
- The performance direction emphasizes increased and faster **processing** power with ever larger word sizes and **addressable** space, such **microprocessors** are used as the **CPUs** in computer systems
- Conversely, however, many **control** applications place far more emphasis on a reduced system footprint (chip count and size) and low power and have no need for increased **processing** power
- These applications make use of **microcontroller** chips with integrated **memory** and **I/O** interface circuits

Microcontrollers and Microcomputers

• Introduction

- **Microcontrollers** are now used almost exclusively in control applications in place of **discrete** logic designs for a number of reasons:
 - Fewer components and **interconnections** required, therefore reduced assembly cost and greater reliability
 - Lower **power** requirements
 - Simpler system testing, evaluation and redesign, **microcontrollers** can simply be **reprogrammed** if problems are discovered
 - As well as allowing for bug fixes, **firmware** upgrades also allow for the addition of new capabilities to systems at any point after manufacture

Microcontrollers and Microcomputers

• Introduction

- In order to acquire a broad understanding of **microprocessor** systems we will be investigating the principles that underlie both of these two main **applications** in some detail
- For this purpose two basic example systems will be investigated over the course of this unit, a generic **68HC11 microcontroller** system and the **IBM 8088** based **AT PC** system
- This lecture will introduce the core **architecture** of the two **microprocessors** these systems are built around, the Freescale **68HC11** and the Intel **8088**

The 68HC11- A Microcontroller

• The 68HC11

- The Freescale **68HC11** (formerly Motorola), despite its age, is still a very commonly used **microcontroller** chip
- It is manufactured using a high-density complementary metal-oxide semiconductor (**HCMOS**) process
- This process produces devices with low **power** requirements and a high tolerance for noisy signals, these are very important factors in **control** applications hence the popularity of the **68HC11**
- Many different varieties of the **68HC11** exist with different levels of integration and packaging, but the same **operating** principles apply to them all

The 68HC11- A Microcontroller

• The 68HC11

- All variants of the **68HC11** are 8-bit **microprocessors** (thus the word size is one byte) with a 16-bit **address** bus (providing 65,536 bytes of **addressable** space)
- For this course the specific **68HC11 microcontroller** we will base our system example off is the **68HC11A8**
- This particular chip integrates the **68HC11 microprocessor** with 8Kb of **ROM**, 512 bytes of **EEPROM**, 256 bytes of **RAM**, five multifunction **I/O ports** and an 8-bit successive approximation analogue-to-digital converter

**The 68HC11-
A Microcontroller**

• **The 68HC11**

– **I/O Ports**

- In the context of a **microprocessor** system a **port** is a set of pins through which digital information may be transmitted to and from the **microprocessor**
- The **68HC11** has five 8-bit **ports** designated **Port A** to **Port E**
- We will examine the function and uses of these **ports** in more detail in a later lecture

– **Address/Data Bus**

- Sometimes a **microcontroller** requires more **memory** or **I/O ports** than are available in the chip itself, this requires external **data** and **address** line connections
- To minimise the required number of pins **time-multiplexing** is used on some of the **68HC11**'s pins so that they can be used both as **I/O ports** and also **data** or **address** lines for this purpose

**The 68HC11-
A Microcontroller**

• **The 68HC11**

– **Operating Mode**

- The **68HC11** has two main **modes** of operation, single-chip mode and expanded multiplexed (or normal expanded) mode
- In expanded multiplexed mode additional **memory** and/or **I/O port** ICs may be connected via the **address** and **data buses**
- **Port B** is used to output the high-order **address** byte $A_8 - A_{15}$ to the external **address bus**
- **Port C** is used for two different functions, it is used both as the output **port** for the low-order byte, $A_0 - A_7$, of the **address** and as an input/output (bidirectional) **port** connecting the **data** signals, $D_0 - D_7$, to the external **data bus**
- N.B. In this configuration an 8-bit **latch** must be placed on the output of **Port C** in order to store the low-order byte of the **address** allowing the **data** byte to be transferred

**The 68HC11-
A Microcontroller**

• **The 68HC11**

– **Control Bus**

- This is the set of **I/O** signals that synchronise the operation of the **68HC11** with the other elements in the expanded system, these signals include the following:
- **R/W** (Read/Write, output) – controls the direction of transfer over the **data bus**
- **AS** (Address Strobe, output) – Enables the external 8-bit **latch** used to store the low-order **address** byte from **Port C**
- **RESET** (bidirectional) – Driven low to initiate a **reset** sequence in the **68HC11** and any external devices connected to this signal; when this signal returns high the levels present at two specific pins, **MODA** and **MODB**, are read and latched, this sets the operating **mode** of the **microcontroller**

**The 68HC11-
A Microcontroller**

• **The 68HC11**

– **Control Bus**

- **XIRQ** and **IRQ** (**Interrupt** requests) – active low signals from one or more external **I/O** devices; used to indicate that service is required from the **microprocessor**, usually to transfer **data** (**XIRQ** is a non-maskable interrupt)
- **E** (Clock) – the **68HC11** generates a single system **clock** called the **E-Clock** that is used both internally and externally; the frequency of this **clock** is determined by a small **crystal** resonating circuit connected to two input pins, **EXTAL** and **XTAL** with four times the desired system **clock** frequency (the maximum frequency of this **crystal** circuit is 8 MHz, which produces an **E-Clock** frequency of 2 MHz)

**The 68HC11-
A Microcontroller**

• **The 68HC11**

– **Memory Map**

- The **68HC11**'s 64K **address space** ranges from 0000h to FFFFh
- All **RAM**, **ROM** and **I/O** devices in the system must be allocated to designated areas in this **address space** (not all of the **address space** will necessarily be used but no two devices should ever occupy the same space)
- The **ROM** and **EEPROM** internal to the **microcontroller** chip have fixed **addresses** in this **address space**, the **RAM** and **I/O address** assignments can be varied using the **CONFIG register**, but will always occupy a certain proportion of the space
- When operating the **microcontroller** in the expanded mode we must ensure that any external **memory** or **I/O** devices are assigned unused **addresses** in the remaining **address space**

Question

- **The 68HC11 has a 16-bit address bus giving access to 65,535 addressable locations, this means ...**

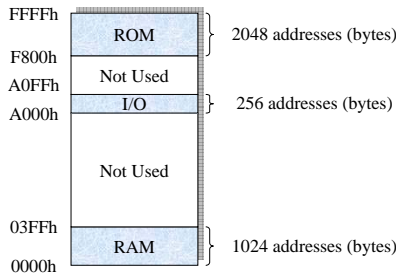
1. The ROM space is 64 Kb
2. The RAM space is 64 Kb
3. Programs can be up to 64 Kb in size
4. Total ROM, RAM and I/O space is up to 64 Kb
5. All of the above

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	31	32									

The 68HC11- A Microcontroller

The 68HC11

Example 68HC11 (Expanded mode) Memory Map



Question

A certain 8-bit microcontroller allocates addresses 0000h – 7FFFh to RAM, 8800h – 88FFh to I/O and C000h – FFFFh to ROM, how much space is available to each?

- 8 Kb RAM, 100 I/O ports and 4 Kb ROM
- 8 Kb RAM, 128 I/O ports and 8 Kb ROM
- 16 Kb RAM, 256 I/O ports and 16 Kb ROM
- 32 Kb RAM, 256 I/O ports and 16 Kb ROM
- 32 Kb RAM, 512 I/O ports and 16 Kb ROM
- 64 Kb RAM, 512 I/O ports and 32 Kb ROM

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	31	32									

The 68HC11- A Microcontroller

The 68HC11

Reset Address Vector

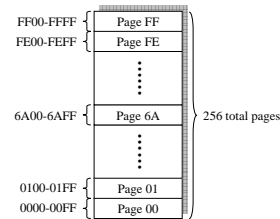
- As part of the initialisation process following a reset, 68HC11 microprocessors always load the contents of memory locations FFEh and FFFFh into the program counter, PC
- These two memory locations always contain the 16-bit address in memory from which the microprocessor will begin executing instructions in response to the reset signal
- This 16-bit value is called the reset address vector and is stored in ROM as it must always be available from the moment power is applied
- Note that the RESET pin must be held low for at least four E-clock cycles in order for the microprocessor to stabilise during a reset

The 68HC11- A Microcontroller

The 68HC11

A Note on Memory Pages

The 64K address space can be divided into 256 blocks or pages of 256 addresses each (each page consists of addresses beginning with the same two hex digits in this case):



In page organisation, each address consists of a page number and a word number:

The 68HC11- A Microcontroller

68HC11 Architecture

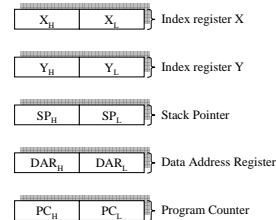
Inside the Microprocessor

- We have briefly described most of the pins and signals that are used to setup and interface to the 68HC11, however, much of the microprocessors internal logic is not directly externally accessible
- These internal elements (for example the program counter) cannot be altered from any of the external pins, we can only interact with them through a program placed in memory, i.e. they are software accessible
- We will now look at these internal elements of the 68HC11 in detail from both a hardware and a programmers perspective, this is usually referred to as the programming model of the microprocessor

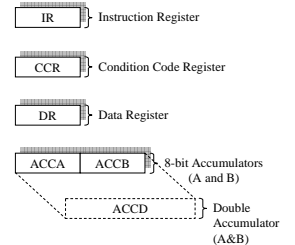
68HC11 Architecture

68HC11 Registers

16-bit Registers



8-bit Registers



• 68HC11 Registers

– Register Functions

- The register structure of different microprocessors varies considerably, however the basic functions they perform is essentially the same – they are used to store data, addresses, instruction op-codes and information relating to the status of microprocessor operations, some may also be used as counters
- We will now examine the functions of each of the 68HC11 registers and particularly how they can be utilised by software

– Instruction Register (8-bits)

- This register is simply used to store op-codes while the instruction decoding circuitry determines how to execute them, it is used automatically during each instruction cycle and is never directly accessed by software

• 68HC11 Registers

– Program Counter (16-bits)

- Always contains the address of the next instruction the microprocessor should fetch and execute, it is automatically incremented after each fetch
- PC_H holds the 8 high-order bits of the address and PC_L holds the 8 low-order bits,
- This separation exists because of the 8-bit word size in the 68HC11, in order to place the contents of this register in memory it must be stored as two consecutive 8-bit segments
- The programmer has no direct access to this register either, although there are many instructions that can be used to indirectly alter its contents, e.g. BRA (unconditional jump)

• 68HC11 Registers

– Data Address Register (16-bits)

- Holds (latches) the address in memory (or I/O space) the microprocessor is currently reading to or writing from

– Accumulators (8 or 16-bit)

- Used to store both operands and results for ALU instructions
- Also has many general purpose uses, e.g. can be used to store data that is being sent to an output device or data received from an input device
- The 68HC11 has two 8-bit accumulators, ACCA and ACCB, which speeds up program execution by allowing two operands to be stored locally (e.g. for addition or subtraction)
- ACCD is a 16-bit accumulator formed by concatenating ACCA and ACCB, it is not a separate register

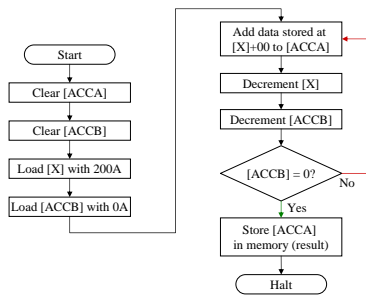
• 68HC11 Registers

– Index Registers (16-bit)

- The 68HC11 has two index registers (X and Y) used for indexed addressing operations (for data arrays)
- In indexed addressing the actual operand address specified in an instruction is the sum of the offset portion of the instruction plus the contents of the index register (e.g. the instruction LDAA 4, X will load ACCA with a byte from the location pointed to by 0004h + [X])
- On the next slide we will look at an example pseudocode program that makes use of indexed addressing to add 10 values stored sequentially in memory from 2001h to 200Ah
- Note that these registers can also be used as general purpose registers for temporary storage or counting functions

• 68HC11 Registers

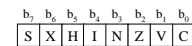
– Indexed Addressing Example



• 68HC11 Registers

– Condition Code Register (8-bit)

- The condition code register (CCR) consists of eight individual flag or status bits used to indicate particular microprocessor conditions:



- C: Carry flag – reflects the carry status of arithmetic operations (reflects MSB carry output of the result), it can also be set or cleared using the instructions SEC or CLC
- V: Overflow flag – set high if an instruction result overflows (out of range), can be cleared or set using CLV or SEV

• 68HC11 Registers

– Condition Code Register (8-bit)

- **Z:** Zero flag – automatically set high if any arithmetic, data transfer or data manipulation operation produces a zero result, automatically cleared to 0 otherwise
- **N:** Negative flag – indicates the sign of the result of the last operation (simply reflects the sign bit)
- **I:** Interrupt mask flag – set high to disable the effect of the interrupt request signal \overline{IRQ} , can be cleared or set using the instructions **CLI** or **SEI** (it is also automatically set or cleared by the microprocessor in some circumstances)
- **H:** Half-carry flag – altered only by addition instructions, set high when an addition produces a carry out from bit 3 (used for BCD operations)

• 68HC11 Registers

– Condition Code Register (8-bit)

- **X:** X Interrupt mask flag – if set high indicates that the non-maskable interrupt request signal, \overline{XIRQ} , is disabled (it cannot be software controlled, only goes high during a reset cycle)
- **S:** Stop Disable flag – set high to disable execution of the STOP instruction, the STOP instruction stops the oscillator and all clocks, causing the microprocessor to enter a very low-power state, this flag is used to prevent accidental execution of this instruction, its state can be altered using the TAP instruction (which performs the operation [ACCA] → [CCR])
- **Note:** The C, V, Z, and N flags are used extensively by the conditional branch instructions, e.g. BEQ (branch if equal to zero) or BCC (branch if carry clear), etc.

• 68HC11 Registers

– Data Register (16-bit)

- The data register is not directly accessible by the programmer, it is used to buffer operands located in memory while they are operated on
- The ALU can operate on operands from either of the accumulators, either index register or a data word in memory (when two operands are involved, one always comes from ACCA or ACCB), the result from the ALU is always sent back to the source of the operand
- If a single operand instruction is carried out on a data word in memory, the result is sent to the data register, from where the microprocessor writes it back into the memory location of the original operand

• Which of the following registers can have its contents specifically altered by a program instruction?

1. The Data Address Register
2. The Program Counter
3. The Instruction Register
4. The Data Register
5. None of the above

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32								

• The Data Address Register ...

1. Holds instruction addresses
2. Holds operand addresses
3. Holds flags
4. Holds op codes
5. Is used in special address modes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32								

• 68HC11 Registers

– Stack Pointer (16-bit)

- This acts as a memory address register used only for access to a special area of memory called the stack, which is used for temporary storage during program execution
- Whenever a word is to be stored to or read from the stack the stack pointer register, SP, is used to determine the appropriate location in memory
- The contents of SP are initialised by the programmer at the beginning of a program, thereafter the SP is automatically decremented after a word is stored on the stack and incremented before a word is read from the stack
- Thus the SP always points to the “top” of the stack

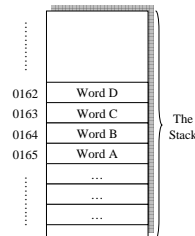
The Stack

Function and Operation

- The stack is an important concept in microprocessor programming so we will examine its operation in a little more detail before continuing
- The stack can be used by the programmer for the temporary storage of register contents and other values, but is also used by the microprocessor for the storage of return addresses and other critical information during subroutine calls and interrupt service routines
- Each time a word is stored in this area of RAM, it is placed in an address location that is one less than the address of the previous word stored, words must be read from the stack in the opposite order from that in which they were placed (hence the stack is referred to as being last in, first out or LIFO)

The Stack

Example



- Here four words have been stored onto the stack, Word A then B then C and finally D
- Initially SP would have pointed to address 0165h
- After Word A was written SP would have been automatically decremented to point to 0164h
- After Word D has been written SP will contain the address 0161h, the new "top" of the stack

The Stack

Example

- Now, if we wish to retrieve Word D from the top of the stack we perform a stack read operation, which will automatically increment SP to 0162h allowing the word stored at that address to be read by the microprocessor
- Similarly a subsequent read will again increment SP allowing the next word down in the stack to be read (Word C in this example), thus only the word at the current top of the stack is ever directly available for reading
- A subsequent stack write will, of course, write back into the same location, thus once a word has been "pulled" from the top of the stack it will be overwritten by any new write to the stack (and SP will then be decremented)

The Stack

Usage

- Special instructions are used to access the stack, e.g. PSHA pushes the contents of ACCA onto the top of the stack and decrements SP, PULA increments SP and then pulls the word off the top of the stack and loads it into ACCA
- The microprocessor automatically controls SP during these operations, all the programmer has to do is to initialise it (and ensure that it never points to locations outside the RAM area reserved for the stack in order to avoid overwriting program or data words)
- The stack can be located anywhere in addressable memory (although it must obviously be in RAM not ROM), its initial value is set using an LDS (Load Stack Pointer) instruction

Question

- Assuming the stack pointer register initially contains the value 6100, what value will it contain after execution of the code to the right?

- 6100
- 6103
- 6104
- 60FD
- 60FC

PSHA
PSHX
PSHB

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	31	32									

Question

- How many READ and WRITE operations will the 68HC11 CPU perform in total when executing the instruction STAA 3D55h?

- 1 READ and 0 WRITE
- 0 READ and 1 WRITE
- 2 READ and 0 WRITE
- 0 READ and 2 WRITE
- 2 READ and 1 WRITE
- 2 READ and 2 WRITE

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	31	32									

The 8088 - A Microcomputer CPU

The 8086/8 CPU

- We will now contrast the 68HC11 microcontroller architecture we have just looked at with that of the 16-bit Intel 8086/88 microprocessor CPU
- The 8088 was a budget version of the 8086, identical in all respects except that it had a cut down 8-bit external data bus instead of the full 16-bit bus present on the 8086
- The popularity of the Intel family of microprocessors was largely ensured in 1981 when IBM chose to use the 8088 as the microprocessor in their new personal computer (which they named the IBM PC)

The 8088 - A Microcomputer CPU

The 8086/8 CPU

- All of Intel's newer microprocessors are still fundamentally based on the original 8086 CPU albeit with a huge number of updates and improvements
- All newer PCs (to date) have been designed to be fully backward compatible with the original 1981 IBM PC design, from a software point of view
- Learning about the 8088 based IBM PC system hence provides us with a fundamental understanding of all current IBM PC compatible computers as well as a general understanding of microcomputer systems

The 8088 - A Microcomputer CPU

The 8088 CPU

- The 8088 microprocessor has a 20-bit address bus providing access to 1 Mbyte ($2^{20} = 1048576$ bytes) of addressable memory (note that although the 8088 is a 16-bit microprocessor, it only has an 8-bit external data bus thus memory is addressed in byte sized chunks)
- Intel microprocessors also use what is known as I/O mapped or isolated I/O addressing in which I/O ports are addressed independently of memory (this will be covered in more detail later), the 8088 supports up to 256 separate I/O ports

The 8088 - A Microcomputer CPU

The 8088 CPU

- Our examination of the 8088 CPU will focus more on the complete microcomputer system than on the specific CPU architecture and as such we will mainly concern ourselves with the programming model
- In this model we are not concerned with every control signal and every register within the microprocessor, we are only interested in the signals and registers that we have access to from a software perspective
- In this model we have access to 1 Mb of addressable memory space, 256 bytes of I/O space and 14 16-bit CPU registers

The 8088 - A Microcomputer CPU

Programming Model of the 8088

Memory Map

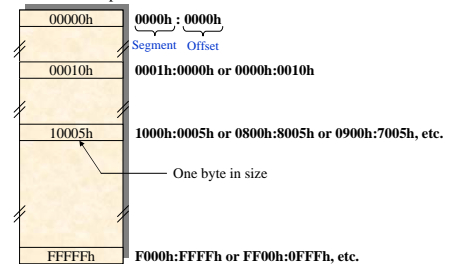
- The 8088 CPU can access up to 1 Mb of memory space, conceptually this can be thought of as a contiguous block from 00000h → FFFFFh,
- In practice, however, this memory is always segmented into 64Kb blocks or pages, i.e. blocks from 0000h → FFFFh
- This was done to maintain compatibility with Intel's earlier 8085 microprocessors, which had only 16 address lines providing access to 64 Kb of memory ($2^{16} = 64K$)
- A logical address combining a 16-bit segment address and a 16-bit offset address (location within the segment) is always used to access a particular 20-bit physical address in memory

The 8088 - A Microcomputer CPU

Programming Model of the 8088

Memory Map

1 Mb Address Space



Question

- Using this addressing scheme, how many different 64Kb segments are locatable in the 8088 address space?

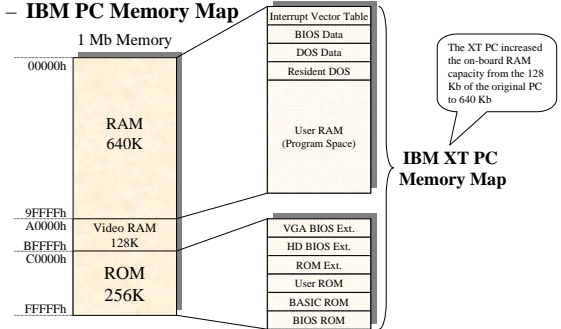
- 8
- 16
- 32
- 128
- 65,536
- 1,048,576

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	31	32									

The 8088 - A Microcomputer CPU

- Programming Model of the 8088

– IBM PC Memory Map



The 8088 - A Microcomputer CPU

- Programming Model of the 8088

– Memory Map – RAM

- Interrupt Vector Table** – this area of RAM contains a table of addresses that point to the locations of service routines that will be called when particular interrupt requests are detected (this will be looked at in more detail in a later lecture)
- BIOS Data** – this area of RAM is used by the BIOS program for temporary storage, it needs this separate area of RAM because the BIOS program is contained in ROM and therefore cannot be written to (data such as the keyboard queue, current screen mode and other temporary variables are stored here)
- DOS Data** – this area of RAM is used by DOS to store temporary information such as the current file-name, directory name, etc.

The 8088 - A Microcomputer CPU

- Programming Model of the 8088

– Memory Map – RAM

- Resident DOS** – This area of RAM contains the main DOS program code, which is always available in memory while the microcomputer is switched on
- User RAM Space** – This is the remaining area of system RAM, about 500 Kb in total, available for users to use for program code, etc.
- Video RAM** – This is an area in addressable space set aside for access to a video display adaptor, video RAM is physically located on the video adaptor but must be mapped into the primary memory address space

The 8088 - A Microcomputer CPU

- Programming Model of the 8088

– Memory Map – ROM

- Extension ROM Space** – this area of ROM is reserved for additional hardware device ROM code, it provides space where routines can be supplied to initialise and control “unknown” hardware
- User ROM Space** – this area of ROM space was provided for users to install their own program code on ROM
- BASIC ROM Space** – the early model IBM PCs had a BASIC boot up program installed on ROM at this location (if another operating system was not found on any of the drives, this ROM was invoked)

The 8088 - A Microcomputer CPU

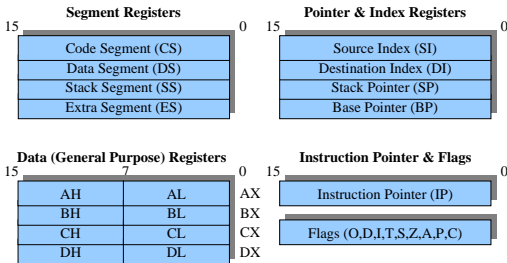
- Programming Model of the 8088

– Memory Map – ROM

- BIOS ROM Space** – The Basic I/O System (BIOS) code is installed via ROM chips at this location, this code initialises the system when it is powered on or reset and also includes various service routines to service interrupts and interface to hardware
- N.B.** The program counter (instruction pointer in Intel speak) will always be initially set to point to the start of BIOS ROM space when power is first applied (or following a reset)

The 8088 - A Microcomputer CPU

• Programming Model of the 8088 – Registers



The 8088 - A Microcomputer CPU

• 8088 Registers

– The Data Registers (AX, BX, CX, DX)

- General purpose registers – for arithmetic calculations, temporary data storage, data transfer, etc.
- The AX (AH | AL) Accumulator Register – the most commonly used register, used in many arithmetic and data transfer operations
- The BX (BH | BL) Base Register – used by several indexed addressing mode instructions to hold the base of a block of data to be manipulated (e.g. an ASCII string)
- The CX (CH | CL) Count Register – used as a counter for a number of instructions (e.g. LOOP, REP)
- The DX (DH | DL) Destination Register – used by I/O (IN, OUT) instructions to hold the address of the port being accessed, also used for 32-bit arithmetic operations

The 8088 - A Microcomputer CPU

• 8088 Registers

– The Instruction Pointer Register (IP)

- This is the 8088s equivalent of the program counter register in the 68HC11
- It is a 16-bit register and thus can only address up to 64K of program space on its own, for this reason it is always used in conjunction with the code segment register (CS) to generate a 20-bit physical address (CS:IP)
- As with the program counter register in the 68HC11 it is not directly accessible by a programmer but can be altered through the use of a number of instructions, e.g. JMP, CALL, INT, etc.

The 8088 - A Microcomputer CPU

• 8088 Registers

– The Segment Registers (CS, DS, SS, ES)

- These registers are required because the total addressable RAM space is $2^{20} = 1 \text{ Mb} = 2^{16+4}$ in size whereas all of the pointer registers are only 16-bits
- The actual physical address in RAM is calculated as: (Segment Register * 16) + Pointer Register, i.e. Segment & Offset (e.g. if CS = 07A0h and IP = 0100h then the physical address = 07A00h + 0100h = 07B00h)
- Note: More than one segment & pointer combination can point to the same physical address (e.g. 07B00 = 07A0:0100 or 07B0:0000)

The 8088 - A Microcomputer CPU

• 8088 Registers

– The Segment Registers (CS, DS, SS, ES)

- Code Segment Register (CS) – used to define the code segment, where the program instructions currently being executed are located
- Data Segment Register (DS) – used to define the data segment, where program data are located
- Stack Segment Register (SS) – used to define the segment used for the stack
- Extra Segment Register (ES) – used to define another segment in addition to the data segment so that memory contents at a distance > 64Kb can be accessed (e.g. direct write to the screen requires that ES be set to point to the screen segment of RAM)

The 8088 - A Microcomputer CPU

• 8088 Registers

– The Segment Registers (CS, DS, SS, ES)

- Note: Due to the internal design of the 8088 a value cannot be loaded directly into a segment register
- To load a value into a segment register the value must first be loaded into one of the data registers (AX, BX, CX or DX) and then transferred into the segment register
- E.g. to load the start of the screen segment into ES:

```
MOV AX, 0B800h
MOV ES, AX
```

**The 8088 - A
Microcomputer CPU**

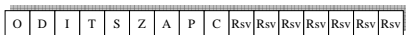
• **8088 Registers**

– **The Stack Pointer and Base Pointer (SP, BP)**

- **Stack Pointer Register (SP)** – basically identical to the **stack** pointer described for the **68HC11**, it always points to the current location of the top of the **stack** in the **stack** segment, (**SS:SP** always points to the top of the **stack**)
- **Base Pointer Register (BP)** – used as a pointer for **memory arrays**, works in the **data** segment (**DS:BP**)

– **The Flags Register (F)**

- Similar to the **68HC11 flags register** except that it is 16-bit, only 9 of the 16 bits are actually used, however, the other 7 bits are reserved



**The 8088 - A
Microcomputer CPU**

• **8088 Registers**

– **The Flags Register (F)**

- **O** = **Overflow flag**, set if signed result is > max value
- **D** = **Direction flag**, used for certain **instructions** (0 = Up)
- **I** = **Interrupt enable flag**
- **T** = **Trap flag**, used for **debugging**
- **S** = **Sign flag**, high for negative 2's complement values
- **Z** = **Zero flag**, set if any result = 0
- **A** = **Auxiliary carry flag**, used for **BCD** calculations
- **P** = **Parity flag**, set if any result has even **parity**
- **C** = **Carry flag**, set if any calculation generates a carry

**The 8088 - A
Microcomputer CPU**

• **8088 Registers**

– **The Index Registers (SI, DI)**

- **Source Index Register (SI)** – used to point to a byte or word in the current **data** segment that needs to be fetched as a part of a block of **data**, this **register** is always used with the **data** segment, i.e. **DS:SI** = physical **address** (e.g. used with string copy **instructions** as the source **address** for **data**)
- **Destination Index Register (DI)** – Similar to **SI**, but used as a destination **address** in the extra segment for a byte or a word that is part of a block of **data** being transferred, i.e. **ES:DI** = physical **address**