538 Lecture Notes Week 5

Announements

- **Midterm: Tuesday, October 25**

Answers to last week's questions

1. With the diagram shown for a port (single bit), what happens if the Direction Register is read? What would you add to the circuit to make this more sensible?

ANSWER:

There are no tri-state buffers activated or clock signals generated when the Direction Register is read. Consequently, nothing is attached to the bus. Usually, the bus wires are connected to Vcc through pull-up resistors so 0xff would be read.

2. Suppose a device has 2 8-bit registers and 2 16-bit registers and that all registers can be read and written. How many RS lines are required?

ANSWER:

*The memory space required is 2*1 + 2*2 = 6 bytes. Thus 3 RS lines are required.*

3. Write code to configure the ATD to sample all channels at (about) 1000 samples/second continuously with 8-bit resolution.

4. Show with a sequence of monitor `wb` (write byte) commands how to clear the LCD display and display the letter 'A' (ascii code: 0x41). (You need to configure PortE and PortS, generate the RS and pulse (E) signals.)

ANSWER:

The LCD controller uses:

- *PortE data: 0x0008*
- *PortE direction:0x0009*
- *PortS data: 0x0248*
- *PortS direction: 0x024A*

First, configure PortE bits 4 and 7 as outputs:

```
>wb $0009 $90
```

Next, configure upper 4 bits of Port S as outputs:

```
>wb $024a $f0
```

Send the command 0x28 (4 bit data, 2 lines) to LCD controller

This requires first setting the LCD E input high (Port E, bit 4):

```
>wb $0008 $10
```

Then, send upper 4 bits of command to DB inputs of controller (Upper half Port S)

```
>wb $0248 $20
```

Pull LCD E input low (Port E, bit 4) to effect transfer:

```
>wb $0008 $00
```

Pull LCD E input high (Port E, bit 4) to set up for next transfer:

```
>wb $0008 $10
```

Then, send lower 4 bits of command to DB inputs of controller (Upper half Port S)

```
>wb $0248 $80
```

Pull LCD E input low (Port E, bit 4) to effect transfer:

```
>wb $0008 $00
```

Send the command 0x0F (display on, cursor on blinking) to LCD controller

This requires first setting the LCD E input high (Port E, bit 4):

```
>wb $0008 $10
```

Then, send upper 4 bits of command to DB inputs of controller (Upper half Port S)

```
>wb $0248 $00
```

Pull LCD E input low (Port E, bit 4) to effect transfer:

```
>wb $0008 $00
```

Pull LCD E input high (Port E, bit 4) to set up for next transfer:

```
>wb $0008 $10
```

Then, send lower 4 bits of command to DB inputs of controller (Upper half Port S)

```
>wb $0248 $f0
```

Pull LCD E input low (Port E, bit 4) to effect transfer:

```
>wb $0008 $00
```

Send the command 0x06 (increment cursor on data write) to LCD controller

This requires first setting the LCD E input high (Port E, bit 4):

```
>wb $0008 $10
```

Then, send upper 4 bits of command to DB inputs of controller (Upper half Port S)

```
>wb $0248 $00
```

Pull LCD E input low (Port E, bit 4) to effect transfer:

```
>wb $0008 $00
```

Pull LCD E input high (Port E, bit 4) to set up for next transfer:

```
>wb $0008 $10
```

Then, send lower 4 bits of command to DB inputs of controller (Upper half Port S)

```
>wb $0248 $60
```

Pull LCD E input low (Port E, bit 4) to effect transfer:

```
>wb $0008 $00
```

Send the character 'A' (0x06) to LCD data register

This requires first setting the LCD E input high (Port E, bit 4) RS (Port E, bit 7):

```
>wb $0008 $90
```

Then, send upper 4 bits of data to DB inputs of controller (Upper half Port S)

```
>wb $0248 $40
```

Pull LCD E input low (Port E, bit 4) keeping RS = 1 to effect transfer:

```
>wb $0008 $80
```

Pull LCD E input high (Port E, bit 4) to set up for next transfer:

```
>wb $0008 $90
```

Then, send lower 4 bits of command to DB inputs of controller (Upper half Port S)

```
>wb $0248 $10
```

Pull LCD E input low (Port E, bit 4) to effect transfer:

```
>wb $0008 $80
```

5. Consider the two ways of waiting for a status register to become "negative". How many bytes of code are required and how many CPU cycles are consumed for each loop?

ANSWER

The number of bytes and clock cycles are given in the comments below.

```
wait:  tst status  ;3 bytes,
       bpl wait    ;2 bytes, 3 cycles if branch else 1 cycle

wait: brclr status,$80,wait ;5 bytes, 5 cycles
```

Both methods require 4 bytes of program memory. The "brclr" takes 5 cycles to loop, 1 more than the 6 cycles needed for the "tst/bpl" method. ***However****, the final loop takes only 4 cycles with the "tst/bpl" method, 1 faster than the other method. Consequently, the "tst/bpl" method will respond faster (by 1 clock cycle) than the "brclr" method; in short, the "tst/bpl" method is superior.*

A/D Conversion Unit continued

- The ATD device on the hcs12 has the following features:
  - 8 analog channels
  - 8 or 10 bit resolution (programmable)
  - More than 100,000 measures/second possible.
- There are 8 16-bit data registers, 1 8-bit data register, 5 8-bit control registers and 2 8-bit status registers as well as reserved test registers. (The device occupies 32 bytes of memory space; i.e. it would have 5 RS lines if it were an external chip.)
- Sufficient detail on the operation is given in Lab 3.
- Complete specs are available in the MC9S12C128V1 data sheet available here.

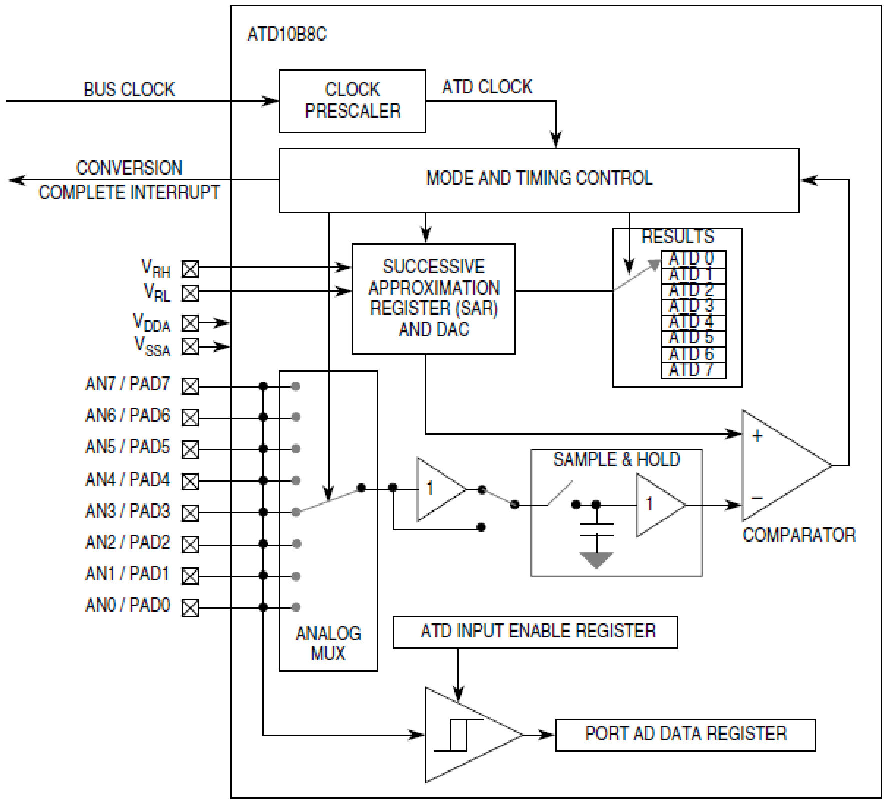
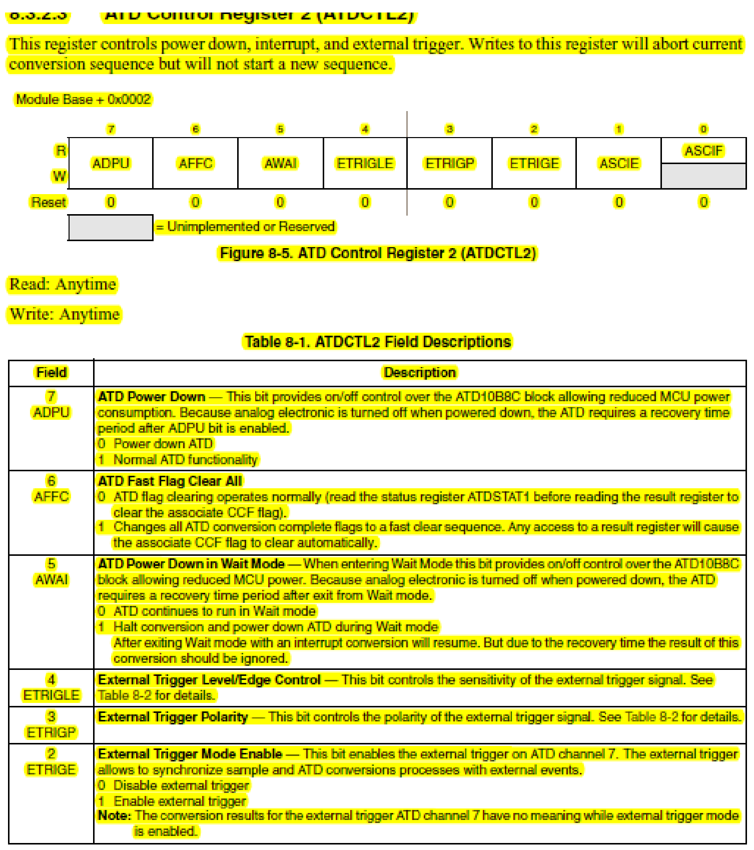


Figure 8-1. ATD10B8C Block Diagram

To use the A/D converter, we:

- Set the control registers for the type of conversion we want including one channel or several, resolution (8 or 10 bits), continuous conversion or one-shot,
- Monitor *status registers* for conversion completion.
- Read the results from the device's *data registers.*



8.3.2.3 ATD Control Register 2 (ATDCTL2)

This register controls power down, interrupt, and external trigger. Writes to this register will abort current conversion sequence but will not start a new sequence.

Module Base + 0x0002

	7	6	5	4	3	2	1	0
R	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIGE	ASCIE	ASCIF
W								▨
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 8-5. ATD Control Register 2 (ATDCTL2)

Read: Anytime

Write: Anytime

Table 8-1. ATDCTL2 Field Descriptions

Field	Description
7 ADPU	**ATD Power Down** — This bit provides on/off control over the ATD10B8C block allowing reduced MCU power consumption. Because analog electronic is turned off when powered down, the ATD requires a recovery time period after ADPU bit is enabled. 0 Power down ATD 1 Normal ATD functionality
6 AFFC	**ATD Fast Flag Clear All** 0 ATD flag clearing operates normally (read the status register ATDSTAT1 before reading the result register to clear the associate CCF flag). 1 Changes all ATD conversion complete flags to a fast clear sequence. Any access to a result register will cause the associate CCF flag to clear automatically.
5 AWAI	**ATD Power Down in Wait Mode** — When entering Wait Mode this bit provides on/off control over the ATD10B8C block allowing reduced MCU power. Because analog electronic is turned off when powered down, the ATD requires a recovery time period after exit from Wait mode. 0 ATD continues to run in Wait mode 1 Halt conversion and power down ATD during Wait mode After exiting Wait mode with an interrupt conversion will resume. But due to the recovery time the result of this conversion should be ignored.
4 ETRIGLE	**External Trigger Level/Edge Control** — This bit controls the sensitivity of the external trigger signal. See Table 8-2 for details.
3 ETRIGP	**External Trigger Polarity** — This bit controls the polarity of the external trigger signal. See Table 8-2 for details.
2 ETRIGE	**External Trigger Mode Enable** — This bit enables the external trigger on ATD channel 7. The external trigger allows to synchronize sample and ATD conversions processes with external events. 0 Disable external trigger 1 Enable external trigger **Note:** The conversion results for the external trigger ATD channel 7 have no meaning while external trigger mode is enabled.

Setting the Control Registers

- First the ADPU (Power, bit 7) bit of ATD Control Register 2 must be set to a 1 to enable the device. (The other bits can be left in their reset state, i.e. 0).
- Program the 4 SC bits in ATD Control Register 3 for the number of channels to convert. (0000 = 1 channel, 0001 = 2 channels...,0111 = all 8 channels, 1xxx also means all channels.)
- Leave the other bits in Control Register 3 as 0.
- Control Register 4 is used to set the resolution (8 or 10 bits) and the conversion timing. We will normally use 8-bit resolution (set bit 7, SRES8, to 1). The default values for the other bits are OK.
- Writing to Control Register 5 initiates a conversion and also indicates single or continuous conversion (bit 5, SCAN), and whether the 16-bit results are left or right justified. The MULT bit (bit 4) is set to 1 for multi-channel conversion.
- Some or all of the input channels can be configured as digital inputs with the ADT Input Enable Register. A "1" indicates that the corresponding input is digital not analog.

The status registers

- The most important bit is bit 7 (SCF) of Status Register 0 which indicates that the conversion is complete on all channels when it is "1". (We will not use the other bits of this register in this course.)
- This bit is set by the ATD hardware. It can be reset by software by:
    - Writing to Control Register 5.
    - By setting the bit to "1" in software!
    - Or, if the AFFC bit in Control Register 2 (bit 6) is set, by reading the result register.
- The second status register (Status Register 1) allows the software to monitor the completion status of individual channels. (We do not use this register in this course.)

The data registers

- The results of conversion can be read in a 16-bit data register associated with each channel (ATDDRHx/ATDDRLx—i.e. high and low bytes of each channel.
- In 8-bit right-justified mode, the result is the lower 8 bits of this register.
- The digital values of any channels configured as digital can be examined in the Port Data Register (PORTAD).

Introduction to Interrupts

- It is very common for an "operation complete" flag to generate an *interrupt*.
- Whether an interrupt is generated is determined by how the programmer configured control registers.
- When an interrupt occurs, program execution is transferred automatically to an *interrupt service routine* (ISR). Upon completion, normal program execution continues from where it was interrupted.
- Interrupts are *ignored* unless the I bit of the Condition Code Register is clear ("0"). This is done with the "cli" instruction. (To disable interrupt recognition, use the "sei" instruction. Interrupts are disabled when the CPU is reset or powered on.)

- When interrupt recognition is enabled and some device causes an interrupt during the execution of an instruction, the interrupt is recognized as soon as the instruction finishes.
- If more than one device simultaneously generates an interrupt, the interrupt with the higher *priority* is recognized and the lower priority interrupts remain pending.
- *All* registers are then pushed onto the stack, the "I" bit in the CCR is set (to disable further recognition of interrupts) and the PC is set to a value specified by the *interrupt vector* associated with the particular interrupt that occurred.
- The PC points to the first instruction of an *Interrupt Service Routine* (ISR) which must:
    - Acknowledge the interrupt. Typically, this is done by clearing the flag in some status register of the device that caused the interrupt.
    - The interrupt is then *serviced*—i.e. operations that should be done are performed by the routine as specified by the programmer.
    - Finally, the ISR "returns from interrupt" with the "rti" instruction. This restores all the registers that were pushed onto the stack when the interrupt was recognized. Significantly, the CCR is restored and consequently the "I" is cleared thus ensuring that further interrupts can be recognized. The PC is also restored and hence the next instruction executed is the one that would normally have been next executed had the interrupt not occured.
    - It is sometimes desirable to re-enable interrupt recognition inside of an ISR by using the "cli" instruction. This is often done if the service routine is long or if any higher priority interrupt may occur. If interrupts are re-enabled inside the ISR, it is **essential** that the interrupt be first acknowledged otherwise an infinite loop occurs.
    - External devices can also generate an interrupt by pulling the IRQ line low. If more than one external device can do this, the ISR must first poll the various external devices to determine which one caused the interrupt and then acknowledge and service it.
    -

Other types of interrupts

- In addition to hardware interrupts coming from devices, the programmer can generate a software interrupt with the "swi" instruction. It always causes an interrupt irrespective of the I bit in the CCR—i.e. it is *non-maskable*.
- Not all bit patterns correspond to a machine code instruction. If the PC ends up pointed to such a bit pattern, an *unimplemented opcode* non-maskable interrupt (also called an *exception* or *trap*) is generated.
- There is also an *non-maksable intrrupt* pin (XIRQ). Such interrupts cannot be ignored. (However, they will be ignored if the X bit in the CCR is "1" and it is "1" following reset. The

programmer can clear it to "0" so that non-maskable interrupts will be recognized but once cleared there is no way to set it back to a "1" short of resetting the CPU.)

The Timer Module (part 1)

- Complete specs are available in the MC9S12C128V1 data sheet available here.

Resets, Clock generation, Interrupts and Operational Modes

- Powering up the CPU, pulling the Reset line low or allowing a Computer Operating Properly (COP) reset will reset the CPU.
- Interrupts are disabled and the CPU begins execution at the address of the Reset Vector (which is in non-volatile memory.)
- Following reset, it is common to set the E clock frequency.
- Details are available http://www.ee.ryerson.ca/~courses/coe538/DataSheets/ClockReset-Chapter9.pdf

Remark on Decimal arithmetic and bcd (packed and unpacked) formats

- In packed bcd format, each hex digit of a byte is in the range 0—9. Hence one byte represents a decimal number between 0 and 99.
- The adda instruction followed by the "daa" (Decimal aAdjust Accumulator A) performs *decimal arithmetic* (of 2 digits).

Example: (Similar to lab 3)

```
            XDEF Entry           ; export 'Entry' symbol
            ABSENTRY Entry       ; for absolute assembly: mark
this as application entry point

    INCLUDE 'derivative.inc'
```

```
                  org RAMStart
  tofCountHundreds         fcb 0   ;Hundreds of counts in packed
bcd
  tofCountTensUnits        fcb 0   ;Tens and units of count in
packed bcd

  ; code section
              ORG   $4000

  Entry:
        jsr init
        cli ; Enable global interrupts
        bra  *



  init
             ldaa #%10000000
             staa TSCR1 ; Enable TCNT by setting bit 7
             staa TFLG2 ; Clear the TOF flag by writing to bit 7
             staa TSCR2  ; Turn timer overflow interrupt on by
setting bit 7
             RTS


  tofISR:
             bset TFLG2,$80

             ldaa tofCountTensUnits
             adda #1
             daa
             staa tofCountTensUnits
             bne isrContinue
             ldaa tofCountHundreds
             adda #1
             daa
             staa tofCountHundreds
  isrContinue:
             rti
  ;***************************************************************
  ;*                 Interrupt Vectors                            *
  ;***************************************************************
              org $ffde
              fdb tofISR
              ORG   $FFFE
              DC.W  Entry           ; Reset Vector
```

Questions

1. Consider the following program:

```
              XDEF Entry            ; export 'Entry' symbol
              ABSENTRY Entry        ; for absolute assembly: mark
this as application entry point
    INCLUDE 'derivative.inc'

              org $400  ;RAM data section
  counter     dc.b $00, $11, $22, $33
              dc.b $44, $55, $66, $77
              dc.b $88, $99, $aa, $bb
              dc.b $CC, $dd, $ee, $ff

              org   $4000 ;ROM code section
  Entry:
              lds   #$0410
              jsr init
              cli

              ldx #$400
              ldy #$401
              ldd 1,x+
              std 2,y+

              bra *   ;Infinite loop (awaiting interrupts)


  tofISR:
             bset TFLG2,$80
             inc counter
             rti

  init:
             ldaa #%10000000
             staa TSCR1 ;Enable TCNT by setting bit 7
             staa TFLG2 ;Clear the TOF flag by writing to bit 7
             staa TSCR2 ;Turn timer overflow interrupt on by
```

```
setting bit 7
             rts

  ;***************************************************************
  ;*                 Interrupt Vectors                           *
  ;***************************************************************
              org $ffde
              dc.w tofISR      ;Timer TOF Vector

              ORG   $FFFE
              DC.W  Entry      ;Reset Vector
```

Vocabulary

New terms are in **bold**.

Memory Dump	The standard way to display the contents of a block of memory in hexadecimal.
CPU	The Central Processor Unit, the computer's heart.
Bus	A set of parellel wires of digital signals
Address Bus	It indicates the address involved in a bus operation
Data Bus	The data being transferred during a bus cycle.
Control Bus	Signals to time and define the type of bus cycle.
IDE	"Integrated Development Environment". Includes editors, compilers, assemblers, disassemblers, etc. We use *CodeWarrior* in this course. In your Java course, you used *Netbeans*. *Eclipse* is another IDE you may use in other courses.
Read Bus Cycle	Data transferred to the CPU.

Write Bus Cycle	Data transferred from the CPU to memory or a memory-mapped device.
Idle Bus Cycle	Bus inactive
Assembler	Software to translate from symbolic assembler to machine code
Disassembler	Software to translate from machine code to symbolic assembler.
Assembler directive	A line of assembler code that gives information to the assembler software and does not correspond to a machine instruction.
Program Counter	A register containing the address of the next instruction.
Stack Pointer	A register containing the address of the top of stack.
Condition Code Register	Contains bits indicating various conditions (such as whether the last number added was zero, negative, generated a carry, etc.) Also called the *Status Register* in some machines (such as Intel processors).
Index Register	A register mainly used as a pointer. It's size equals the width of the Address Bus.
Arithmetic Shift	Only applies to a right shift where the sign bit is "shifted in" from the right maintaining the sign of the shifted value.
Indexed Addressing	Accessing the contents of memory whose address is calculated by adding an offset (usually zero) to an index register.
Indirect Indexed Addressing	Using indexed addressing to obtain another pointer in memory and then dereferencing the location it points to.
Overflow (V) bit	Set when the result of an addition/subtraction will not fit in the number of bits used.
Effective Address	The address that will be used in indexed addressing. i.e. the index register + offset.
Addressing Mode	The way an operand is specified.
Inherent Addressing	The operand is inherent in the instruction itself.
Immediate Addressing	The operand is part of the instruction (a specific field) and no further memory reference is required.
PC-Relative Addressing	The operand is an offset relative to the current value of the PC.
Subroutine	Similar to a C function. Parameters, if any, can be passed in registers or on the Stack.
Side effect	A change in the CPUs state (such as a register) unrelated to the subroutine's function.
Parameter	An argument passed to a subroutine. They are usually passed in registers or on the stack.
Direction Register	A control register associated with a parallel port that configures individual bits to be inputs or outputs.

Memory Mapped Device	A peripheral device whose internal registers are mapped to specific memory locations.

Questions

1. Consider the following program:

```
              XDEF Entry            ; export 'Entry' symbol
              ABSENTRY Entry        ; for absolute assembly: mark
this as application entry point
    INCLUDE 'derivative.inc'

              org $400  ;RAM data section
  counter     dc.b $00, $11, $22, $33
              dc.b $44, $55, $66, $77
              dc.b $88, $99, $aa, $bb
              dc.b $CC, $dd, $ee, $ff

              org   $4000 ;ROM code section
  Entry:
              lds   #$0410
              jsr init
              cli

              ldx #$400
              ldy #$401
              ldd 1,x+
              std 2,y+

              bra *   ;Infinite loop (awaiting interrupts)


  tofISR:
             bset TFLG2,$80
             inc counter
             rti

  init:
```

```
             ldaa #%10000000
             staa TSCR1 ;Enable TCNT by setting bit 7
             staa TFLG2 ;Clear the TOF flag by writing to bit 7
             staa TSCR2 ;Turn timer overflow interrupt on by
setting bit 7
             rts

  ;**************************************************************
  ;*                 Interrupt Vectors                           *
  ;**************************************************************
              org $ffde
              dc.w tofISR      ;Timer TOF Vector

              ORG   $FFFE
              DC.W  Entry      ;Reset Vector
```

Show the memory dump 0x0400-0x040f just before the “rti” instruction is executed. Assume that the interrupt occcurs during the “bra *” instruction.

ANSWER:

```
0400 01 00 11 33 44 55 66 C0 11 00 04 01 04 03 40 12
```

Vocabulary

New terms are in **bold**.

Memory Dump	The standard way to display the contents of a block of memory in hexadecimal.
CPU	The Central Processor Unit, the computer's heart.
Bus	A set of parellel wires of digital signals
Address Bus	It indicates the address involved in a bus operation

Data Bus	The data being transferred during a bus cycle.
Control Bus	Signals to time and define the type of bus cycle.
IDE	"Integrated Development Environment". Includes editors, compilers, assemblers, disassemblers, etc. We use *CodeWarrior* in this course. In your Java course, you used *Netbeans*. *Eclipse* is another IDE you may use in other courses.
Read Bus Cycle	Data transferred to the CPU.
Write Bus Cycle	Data transferred from the CPU to memory or a memory-mapped device.
Idle Bus Cycle	Bus inactive
Assembler	Software to translate from symbolic assembler to machine code
Disassembler	Software to translate from machine code to symbolic assembler.
Assembler directive	A line of assembler code that gives information to the assembler software and does not correspond to a machine instruction.
Program Counter	A register containing the address of the next instruction.
Stack Pointer	A register containing the address of the top of stack.
Condition Code Register	Contains bits indicating various conditions (such as whether the last number added was zero, negative, generated a carry, etc.) Also called the *Status Register* in some machines (such as Intel processors).
Index Register	A register mainly used as a pointer. It's size equals the width of the Address Bus.
Arithmetic Shift	Only applies to a right shift where the sign bit is "shifted in" from the right maintaining the sign of the shifted value.
Indexed Addressing	Accessing the contents of memory whose address is calculated by adding an offset (usually zero) to an index register.
Indirect Indexed Addressing	Using indexed addressing to obtain another pointer in memory and then dereferencing the location it points to.
Overflow (V) bit	Set when the result of an addition/subtraction will not fit in the number of bits used.
Effective Address	The address that will be used in indexed addressing. i.e. the index register + offset.
Addressing Mode	The way an operand is specified.
Inherent Addressing	The operand is inherent in the instruction itself.
Immediate Addressing	The operand is part of the instruction (a specific field) and no further memory reference is required.
PC-Relative Addressing	The operand is an offset relative to the current value of the PC.

Subroutine	Similar to a C function. Parameters, if any, can be passed in registers or on the Stack.
Side effect	A change in the CPUs state (such as a register) unrelated to the subroutine's function.
Parameter	An argument passed to a subroutine. They are usually passed in registers or on the stack.
Direction Register	A control register associated with a parallel port that configures individual bits to be inputs or outputs.
Memory Mapped Device	A peripheral device whose internal registers are mapped to specific memory locations.

Bcd ascii

Bcd ascii