Micro computer Organization

1 Base
Basic Components

- CPU
- System Memory
  - Data
  - Program
- System Buses
- Central Processing Unit
- Power & Timing Support
- Input-Output Subsystem
- VDD
- CLK
- RESET
MPU vs MCU

• Microprocessor Unit (MPU)
  – CPU (called Microprocessor) is a die by itself
  – All other components external to die
    • Basically on one or several boards
  – CPU is “optimized”

• Microcontroller Unit (MCU)
  – All components in a die
  – Less complex
MCU: Generic structure
Practical examples: MSP430G2x01
MSP430G2x53 (Launchpad)

NOTE: Port P3 is available on 28-pin and 32-pin devices only.
Practical Examples: MSP430F5437
RISC vs CISC (1/2)

• RISC (Reduced Instruction Set Computer) architecture:
  – Small set of instructions (optimized)
  – Emphasis on simpler hardware
  – Many instructions take one cycle only
  – Larger number of code lines
  – Relatively large number of CPU registers to minimize interaction with memory
RISC vs CISC (2/2)

- **CISC** (Complex Instruction Set Computer) architecture:
  - Large set of instructions
  - Emphasis on simpler software
  - Instructions are of different clock cycles.
  - Smaller number of code lines

- **RISC** has become more popular, but final decision depends on needs and other considerations.
Hardware Vs Programmers Model

• Hardware model focuses on hardware characteristics that supports instructions, timing, etc.

• Programmer’s model focuses on
  – Instructions and addressing mode syntax
  – Memory and IOmap
  – Transfers, etc.
  – Program models of IO registers

• In embedded systems, both models are needed, at least at system level
Microcomputer Organization

II CPU
CPU Components

• Hardware components
  – Control Unit (CU)
  – Registers
  – Arithmetic Logic Unit (ALU)
  – Bus Interface Logic unit (BIL)

• Software Components
  – Instruction Set
  – Addressing modes
**Data Path and Control Path**

- **Data Path:** HW components used to perform operations
  - ALU
  - Registers and Internal Buses
  - Specialized units

- **Control Path:** HW components controlling system operation
  - CU
  - BIL
  - Timing and synchronization units
ALU (Arithmetic Logic Unit):

• Combinatorial circuit which realizes the arithmetic, logic, and other CPU operations.

• The width of ALU operands gives name to the classification in bits of the MCU:
  – 4-bit microcontrollers, 16-bit microcontrollers, etc.

• ALU controls several flags in Status Register
CU (Control Unit):

• Sequential circuit – finite state machine – that controls the activity of the system,
  – Retrieves instructions from memory
  – Coordinates the instruction cycle
  – Coordinates transfers
  – Etc.
CPU: Registers

• **Special Purpose Registers** used for specific operation. Common ones are
  – Instruction Register (IR) – not available to programmer -
  – Program Counter (PC)
  – Stack Pointer (SP)
  – Status Register (SR)

• General Purpose registers

• “Invisible registers”, for internal operation, not available to programmer
Instruction Cycle or CPU Cycle: Fetch-Decode Execute

- **Fetch:** The CU brings a new instruction from memory through BIL
  - Register PC provides the address of instruction to be fetched
  - Instruction is stored in IR
- **Decode:** instruction meaning is deciphered
- **Execute:** CU commands the corresponding units to perform the actions.
- **Reset:** A defined state after power up or after a reset occurs
Important Note: Instruction Address in PC

A. Register PC **always** has the address of the following instruction after the execution phase.

B. If the execution phase does not change contents of PC, then **the address of the following instruction is in PC after the decoding phase.**
Just after a previous cycle:

a) Contents of IR is irrelevant.

b) PC points to next instruction
CU puts PC contents in Address Bus (using BIL unit) and reads (using Data Bus and Control Bus) memory contents and puts result into Instruction Register (IR)

PC increases its value pointing to next address.

(Fetched word in this first movement is the Instruction Word)
CU decodes:

**Add contents of R10 to contents of R6**

The information is complete, so decoding is finished.

a) In Register Transfer Notation (RTN):

\[ R6 \leftarrow R6 + R10 \]

b) Decoding is finished and PC is pointing to address following this instruction

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F812</td>
<td>4809</td>
</tr>
<tr>
<td>0F810</td>
<td>2C07</td>
</tr>
<tr>
<td>0F80E</td>
<td>2038</td>
</tr>
<tr>
<td>0F80C</td>
<td>2034</td>
</tr>
<tr>
<td>0F80A</td>
<td>5292</td>
</tr>
<tr>
<td>0F808</td>
<td>45AF</td>
</tr>
<tr>
<td>0F806</td>
<td>403A</td>
</tr>
<tr>
<td>0F804</td>
<td>5A06</td>
</tr>
<tr>
<td>2038</td>
<td>D54A</td>
</tr>
<tr>
<td>2036</td>
<td>6D45</td>
</tr>
<tr>
<td>2034</td>
<td>5FA8</td>
</tr>
</tbody>
</table>

R6 = C3D0

**C = 1**  **Z = 0**  **N = 0**  **V = 1**
The processor executes what decoding indicated:

a) Old contents of destination is lost and has been replaced with new result

b) Flags have been affected by this instruction

c) IR contents is the same, but it is irrelevant
CU fetches Instruction word and increments PC which is now pointing to next address.

CU fetches Instruction word and increments PC which is now pointing to next address.
CU determines that the instruction needs the data in memory after the instruction word, so it is necessary to fetch this word (Not an instruction word) to complete decoding.

Therefore, it will fetch the word and place it on the IR before completing decoding.

PC is incremented accordingly.
### DECODE (2)

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F812</td>
<td>4809</td>
</tr>
<tr>
<td>0F810</td>
<td>2C07</td>
</tr>
<tr>
<td>0F80E</td>
<td>2038</td>
</tr>
<tr>
<td>0F80C</td>
<td>2034</td>
</tr>
<tr>
<td>0F80A</td>
<td>5292</td>
</tr>
<tr>
<td>0F808</td>
<td>45AF</td>
</tr>
<tr>
<td>0F806</td>
<td>403A</td>
</tr>
<tr>
<td>0F804</td>
<td>5A06</td>
</tr>
</tbody>
</table>

**DECODED instruction:**

Copy (move) the word 45AF into R10

**a)** In RTN:

R10 ← 45AF  
Also  
R10 ← #45AF

**b)** Decoding is finished and PC is pointing to address following this instruction
The processor executes what decoding indicated:

a) Old contents of destination is lost and has been replaced with new result

b) Flags are not affected by this instruction

c) IR contents is the same, but it is irrelevant
CU fetches *Instruction word* and increments PC which is now pointing to next address.

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F812</td>
<td>4809</td>
</tr>
<tr>
<td>0F810</td>
<td>2C07</td>
</tr>
<tr>
<td>0F80E</td>
<td>2038</td>
</tr>
<tr>
<td>0F80C</td>
<td>2034</td>
</tr>
<tr>
<td>0F80A</td>
<td>5292</td>
</tr>
<tr>
<td>0F808</td>
<td>45AF</td>
</tr>
<tr>
<td>0F806</td>
<td>403A</td>
</tr>
<tr>
<td>0F804</td>
<td>5A06</td>
</tr>
<tr>
<td>2038</td>
<td>D54A</td>
</tr>
<tr>
<td>2036</td>
<td>6D45</td>
</tr>
<tr>
<td>2034</td>
<td>5FA8</td>
</tr>
</tbody>
</table>

CU fetches **Instruction word** and increments PC which is now pointing to next address.
### Decode (1)

### Address | Contents
---|---
0F812 | 4809
0F810 | 2C07
0F80E | 2038
0F80C | 2034
0F80A | 5292
0F808 | 45AF
0F806 | 403A
0F804 | 5A06
2038 | D54A
2036 | 6D45
2034 | 5FA8

IR: 5292 45AF XXXX

CU determines that the instruction needs data in memory after the instruction word. This time, two words, to complete decoding.

Therefore, it will fetch the words and place them on the IR.

PC is incremented accordingly.

C = 1  Z = 0
N = 0  V = 0
# Decode (2,3)

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F812</td>
<td>4809</td>
</tr>
<tr>
<td>0F810</td>
<td>2C07</td>
</tr>
<tr>
<td>0F80E</td>
<td>2038</td>
</tr>
<tr>
<td>0F80C</td>
<td>2034</td>
</tr>
<tr>
<td>0F80A</td>
<td>5292</td>
</tr>
<tr>
<td>0F808</td>
<td>45AF</td>
</tr>
<tr>
<td>0F806</td>
<td>403A</td>
</tr>
<tr>
<td>0F804</td>
<td>5A06</td>
</tr>
<tr>
<td>2038</td>
<td>D54A</td>
</tr>
<tr>
<td>2036</td>
<td>6D45</td>
</tr>
<tr>
<td>2034</td>
<td>5FA8</td>
</tr>
</tbody>
</table>

DECODED instruction:

Add the word in memory with address 2034 to the word in memory with address 2038

a) In RTN:

\[(2038) \leftarrow (2034) + (2038)\]

Also

\[&2038 \leftarrow &2034 + &2038\]

b) Decoding is finished and PC is pointing to address following this instruction

C = 1    Z = 0
N = 0    V = 1
The processor executes what decoding indicated:

a) Old contents of destination is lost and has been replaced with new result

b) Flags are affected by this instruction

c) IR contents is the same, but it is irrelevant
CU fetches Instruction word and increments PC which is now pointing to next address.
CU decodes:

IF flag C is set (C=1)
THEN go to instruction at address F820h

Technically, if C is set then
PC ← PC + 2 (0007h)

a) In RTN, express objective;
If C=1, GOTO to F820h or
IF C=1, PC ← F820h

b) Decoding is finished and
PC is pointing to address following this instruction
The execution in this case changes the contents of the PC.

This will cause a JUMP in the sequence of instructions.

The next instruction to be fetched is not the one after the current one.

The instruction does not affect flags.
Important facts to remember

• Instruction can have one or more words
  – **Instruction word:** First word in the set.
  – **Instruction word:** Op Code and Addressing modes

• After the execution state, the PC has the address of the next instruction

• After the decode state, the PC holds the memory address after the current instruction
  – Execution of Program flow instructions may alter PC
  – For other instructions, this is the address of next instruction
Status Register (SR)

• Contains flags related to result of execution for some instructions involving ALU and a control Interrupt Flag. All systems include
  – Carry Flag (C)       Zero Flag (Z)
  – Negative Flag (N)    Overflow Flag (V)
  – Interrupt flag (IF) or General Interrupt flag (GIE)
    • Interrupts blocked with IF are called maskable

• Contains group of bits related to system control
What C, Z, N, V flags mean in addition and subtraction: illustration

- **Carry flag** = Cout
- **Zero flag** = 1 if $S_3=S_2=S_1=S_0 = 0$
  = 0 otherwise
- **N flag** = $S_3$
- **V flag** = 1 if $A_3 = B_3 \neq S_3$
  = 0 otherwise

This explanation serves addition and subtraction only. Other operations will yield flags as explained in user guide, and may differ from CPU to CPU models.
Carry flag: Special remarks

• In arithmetic subtraction and addition operations, the Carry Flag may have dual function: Carry and Borrow
  – Some MCU’s have a separate borrow flag or else
    Carry Flag = M ⊕ Cout (applied to previous illustration)

• Depending on the MCU model (see user guide):
  – C=1  if a borrow is needed in subtraction ... or
  – C=0  if a borrow is needed in subtraction
    • MSP430 adheres to this convention
Flags and Number comparison (Using A-B)

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Unsigned Numbers</th>
<th>Signed Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A = B$</td>
<td>$Z=1$</td>
<td>$Z=1$</td>
</tr>
<tr>
<td>$A \neq B$</td>
<td>$Z=0$</td>
<td>$Z=0$</td>
</tr>
<tr>
<td>$A \geq B$</td>
<td>$C=1$</td>
<td>$N=V$</td>
</tr>
<tr>
<td>$A &gt; B$</td>
<td>$C=1$ and $Z=0$</td>
<td>$N=V$ and $Z=0$</td>
</tr>
<tr>
<td>$A &lt; B$</td>
<td>$C=0$</td>
<td>$N\neq V$</td>
</tr>
<tr>
<td>$A \leq B$</td>
<td>$C=0$ or $Z=1$</td>
<td>$N\neq V$ or $Z=1$</td>
</tr>
</tbody>
</table>

**Note:** This table assumes that $C=0$ indicates need of borrow in subtraction
Stack Pointer (SP)

• Manages a particular memory segment called STACK

• STACK operations are “PUSH” (Store) and “POP” (Retrieve)
  – The SP register contents indicates where data is stored in a push operation, and where data is retrieved from in a pop operation

• SP contents is usually named “Top of Stack”
  – Details later.
Stack features

• The Stack may be
  – Defined by user (usual case) by initializing SP, or
  – hardware defined (some MCU models)

• The stack serves the user to temporary store data, and temporary free a register for some use.

• The stack and the SP register also support in the background special activities that require saving data temporarily. Example:
  – Program flow transfers to and from subroutines (function)
  – Management of Interrupt service.
Microcomputer Organization

III Hardware Characteristics of MSP430 CPU - CPUX
• MSP430 offers two architectures:
  – Original MSP430 64K memory, with CPU
  – Extended MSP430X with 1M memory capacity, CPUX

• MSP430X is 100% downward compatible with MSP430

• ALU
  – CPU: 16 bits
  – CPUX: 20 bits
CPU
CPU and CPUX registers

• 16 registers.
  – CPU has 16-bit registers
  – CPUX has 20-bit registers that operate as CPU registers for all CPU instructions.
  – Status Register has 16-bits in both cases.
• Register R0: Program Counter (PC) with bit0=0, hardwired
• Register R1: Stack Pointer (SP) with Bit0 = 0, hardwired
• Register R2: Status Register (SR), 16-bits only
  – Also works as constant generator (CG1)
• Register R3: Constant Generator (CG2) (actually not a register)
• Registers R4 to R15: General Purpose registers.
MSP430 Status Register (1/2)

- C: Carry Flag
- Z: Zero Flag
- N: Sign Flag
- GIE: Global Interrupt enable Flag
- V: Overflow Flag
MSP430 Status Register (2/2)

• CPU Off: Turns on and off the CPU
  – CPU Off if CPUOFF=1

• OSCOFF: Turns on and off the Crystal Oscillator
  – Oscillator Off when OSCOFF=1

• SCG1 and SCG0 are combined with CPUOFF and OSCOFF to define the modes of operation
Modes of operation

<table>
<thead>
<tr>
<th>SCG1</th>
<th>SCG0</th>
<th>OSCOFF</th>
<th>CPUOFF</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Active</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>LPM0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>LPM1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>LPM2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>LPM3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>LPM4</td>
</tr>
</tbody>
</table>

LPM: Low power mode