PIC32

Lembit Jürimägi

BASYS MX3 Board

- Built around a PIC32MX370F512L chip
- Designed for teaching embedded systems
- Can be used with MPLABX IDE
- Connects various peripherals like:
  - LEDs
  - switches
  - buttons
  - 7-segment display
  - LCD character display
  - microphone
  - speaker
  - etc

PIC32MX3 series chipsets

- Use MIPS32 microprocessor core
- Have many multifunctional pins that can be configured as:
  - digital input
  - digital output
  - analog input
  - external timer
  - external interrupt
  - etc
- Pins are mapped to bits of ports PORTA – PORTG
- Various registers as ANSELx, TRISx, etc are used to configure pins
- Reading and writing through PORTx and LATx registers
- Registers are mapped to memory addresses that MIPS32 core can access

Devices mapped to ports

- We will be using a very small subset of features:

LD0	PORTA bit0	SW0	PORTF bit3
LD1	PORTA bit1	SW1	PORTF bit5
LD2	PORTA bit2	SW2	PORTF bit4
LD3	PORTA bit3	SW3	PORTD bit15
LD4	PORTA bit4	SW4	PORTD bit14
LD5	PORTA bit5	SW5	PORTB bit11
LD6	PORTA bit6	SW6	PORTB bit10
LD7	PORTA bit7	SW7	PORTB bit9
R	PORTD bit2		
G	PORTD bit12		
B	PORTD bit3		

Ports

- Ports can work as analog input, digital input, digital output
- Analog output can be realized with pulse-width modulation (PWM)
- Ports must be configured for use
- To digitally write to a port either set or clear the corresponding LATxSET or LATxCLR memory location
- To read from a port access the PORTx memory location

Configuring Ports

- To configure a pin as digital the corresponsing ANSELx bit needs to be cleared
  - ANSELACLR = 0x1; // for LED0
- To configure a pin as output the corresponding TRISx bit needs to be cleared
  - TRISACLR = 0x1; // for LED0
- To configure a pin as input the corresponding TRISx bit needs to be set
  - TRISFSET = 0x8; // for SW0

MIPS32 instructions

- All instructions are 32 bits in length
- Memory addresses are 32 bits
- Processor has 32 registers
  - register $0 is hardwired to contain value 0 and cannot be written to
  - register $1 is reserved
  - registers $2 to $23 are free to use
  - registers $28 to $31 used for stack pointer, return address etc
- Instructions are usually in format
  - operation, destination, source1, source2
  - operation, destination, 16 bit immediate value
- Therefore using 32 bit constants and memory addresses takes 2 instructions

MIPS32 instructions

- We will be using very small subset of instructions:

lui reg, base	load base address to reg
lw reg,offset(reg)	load 32-bit word from memory at address base + offset to reg
sw reg,offset(reg)	store reg to memory at address base + offset
li reg,imm	load 32-bit immediate to reg
addi regd,imm	ADD 16-bit immediate to regd and store in regd
andi regd,regs,imm	AND regs with 16-bit immediate and store in regd
beq regs,regd,label	compare regs and regd, if equal jump to label
bne regs,regd,label	compare regs and regd, if not equal jump to label
j label	jump unconditionally to label
nop	no operation, needed after every jump and branch instruction

MIPS32 instructions

sll regd,regs,sa	shift regs sa bits to the left and store in regd
sllv regd,regs,rega	shift regs the value of rega bits to the left and store in regd
srl regd,regs,sa	shift regs sa bits to the right and store in regd
srlv regd,regs,rega	shift regs the value of rega bits to the right and store in regd

- If you need others then refer to the MIPS32 instruction manual

MPLABX

- Open MPLAB X IDE under Programming
- File -> New Project
  - Microchip Embedded > Standalone Project
    - Next
  - Family: 32bit MCUs
  - Device: PIC32MX370F512L
    - Next
  - Tool: Other Tools > Licensed Debugger > BASYS MX3
    - Next
  - Compiler Toolchain: XC32 > XC32 (v1.44)
    - Next
  - Project Name and Folder
    - Make sure to save the project on drive P, default location is home folder
    - Finish

MPLABX

- Right-click Source Files -> New -> Other
  - Categories: Assembler > AssemblyFile.s
- Copy the contents of example.s
- Click the button Make and Program Device

- Once you have built the interpreter you can output the generated assembly code directly to MPLAB:

```
./pic example.pic ./labor2.X/prog.s
```

The language of practice 2

- We are using only a part of the BASYS board that has leds and switches.
- The question becomes what sort of language should we make for controlling this functionality.
- It could be a general purpose language with loops, conditions etc.
- But it could also be a domain specific language that hides all of that from the user.
- This is for you to decide.