به نام خدا

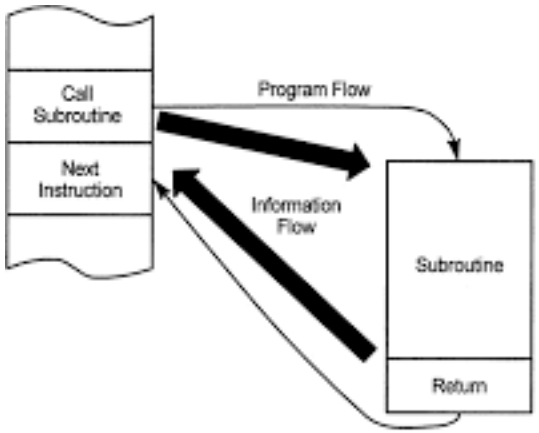
آشنایی با زبان اسمبلی AVR

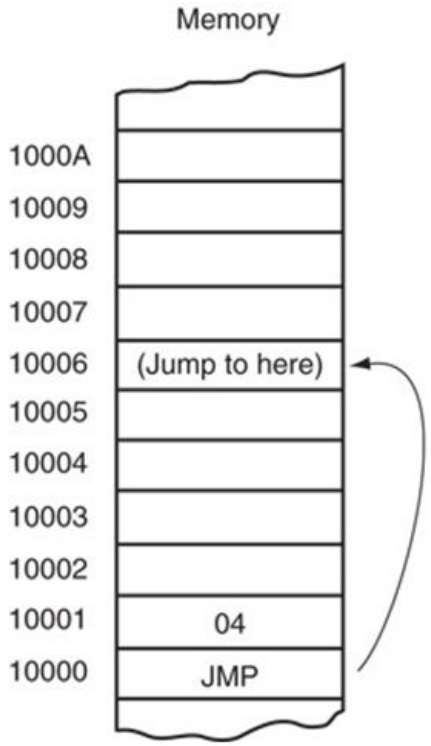
زیرروال‌ها

Dr. Aref Karimiafshar
A.karimiafshar@ec.iut.ac.ir

دستورات پرش

- In the sequence of instruction to be executed, it is often necessary to transfer program control to a different location.
- Instruction in AVR to achieve this:
  - Branches (JUMP)
  - CALL





فراخوانی زیرروال (CALL)

- Is used to call a subroutine
- Subroutines
  - To perform tasks that need to be performed frequently
- Advantages
  - Makes a program more structured
  - Saving memory space

فراخوانی زیرروال (CALL)

- Four instructions for call subroutine:
  - CALL (long call)
  - RCALL (relative call)
  - ICALL (indirect call to Z)
  - EICALL (extended indirect call to Z)

- The choice of which one to use depends on the target address

CALL – Long Call to a Subroutine

- Calls to a subroutine within the entire Program memory. The return address (to the instruction after the CALL) will be stored onto the Stack. The Stack Pointer uses a post-decrement scheme during CALL.

Operation:

(i) PC ← k Devices with 16-bit PC, 128KB Program memory maximum.

(ii) PC ← k Devices with 22-bit PC, 8MB Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	CALL k	0 ≤ k < 64K	PC ← k	STACK ← PC+2 SP ← SP-2, (2 bytes, 16 bits)
(ii)	CALL k	0 ≤ k < 4M	PC ← k	STACK ← PC+2 SP ← SP-3 (3 bytes, 22 bits)

CALL – Long Call to a Subroutine

32-bit Opcode:

1001	010k	kkkk	111k
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words 2 (4 bytes)

Cycles 4 devices with 16-bit PC

5 devices with 22-bit PC

RCALL – Relative Call to Subroutine

- Relative call to an address within PC - 2K + 1 and PC + 2K (words). The return address (the instruction after the RCALL) is stored onto the Stack. For AVR microcontrollers with Program memory not exceeding 4K words (8KB) this instruction can address the entire memory from every address location. The Stack Pointer uses a post-decrement scheme during RCALL.

	Operation:	Comment:		
(i)	PC ← PC + k + 1	Devices with 16-bit PC, 128KB Program memory maximum.		
(ii)	PC ← PC + k + 1	Devices with 22-bit PC, 8MB Program memory maximum.		
	Syntax:	Operands:	Program Counter:	Stack:
(i)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1 SP ← SP - 2 (2 bytes, 16 bits)
(ii)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1 SP ← SP - 3 (3 bytes, 22 bits)

RCALL – Relative Call to Subroutine

16-bit Opcode:

1101	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	3 devices with 16-bit PC
	4 devices with 22-bit PC

ICALL – Indirect Call to Subroutine

- Calls to a subroutine within the entire 4M (words) Program memory. The return address (to the instruction after the CALL) will be stored onto the Stack. The Stack Pointer uses a post-decrement scheme during CALL.

	Operation:	Comment:
(i)	PC(15:0) ← Z(15:0)	Devices with 16-bit PC, 128KB Program memory maximum.
(ii)	PC(15:0) ← Z(15:0) PC(21:16) ← 0	Devices with 22-bit PC, 8MB Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	ICALL	None	See Operation	STACK ← PC + 1 SP ← SP - 2 (2 bytes, 16 bits)
(ii)	ICALL	None	See Operation	STACK ← PC + 1 SP ← SP - 3 (3 bytes, 22 bits)

ICALL – Indirect Call to Subroutine

16-bit Opcode:

1001	0101	0000	1001

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	3 devices with 16-bit PC
	4 devices with 22-bit PC

EICALL – Extended Indirect Call to Subroutine

- Indirect call of a subroutine pointed to by the Z (16 bits) Pointer Register in the Register File and the EIND Register in the I/O space. This instruction allows for indirect calls to the entire 4M (words) Program memory space. The Stack Pointer uses a post-decrement scheme during EICALL.

Operation:

(i) PC(15:0) ← Z(15:0)

PC(21:16) ← EIND

	Syntax:	Operands:	Program Counter:	Stack:
(i)	EICALL	None	See Operation	STACK ← PC + 1 SP ← SP - 3 (3 bytes, 22 bits)

EICALL – Extended Indirect Call to Subroutine

16-bit Opcode:

1001	0101	0001	1001

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words 1 (2 bytes)

Cycles 4 (only implemented in devices with 22-bit PC)

RET – Return from Subroutine

- Every subroutine needs RET as the last instruction.
- Returns from subroutine. The return address is loaded from the STACK. The Stack Pointer uses a pre-increment scheme during RET.

	Operation:	Comment:
(i)	PC(15:0) ← STACK	Devices with 16-bit PC, 128KB Program memory maximum.
(ii)	PC(21:0) ← STACK	Devices with 22-bit PC, 8MB Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RET	None	See Operation	SP ← SP + 2, (2 bytes, 16 bits)
(ii)	RET	None	See Operation	SP ← SP + 3, (3 bytes, 22 bits)

RET – Return from Subroutine

16-bit Opcode:

1001	0101	0000	1000

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	4 devices with 16-bit PC
	5 devices with 22-bit PC

Stack

- Stack is a section of RAM used by CPU to store information temporarily
  - Data
  - Address
- CPU needs this storage area because there are only a limited number of registers
- There is a register in CPU to point to stack
  - Called Stack Pointer (SP) register
    - Two register in I/O memory space
      - SPL: Low byte of SP
      - SPH: High byte of SP

Operation on the Stack

- PUSH
  - Storing information on the Stack
    - SP points to top of the stack (TOS)
    - As we push data, the data are saved where SP points
    - SP is decrement by one

	Syntax:	Operands:	Program Counter:	Stack:
(i)	PUSH Rr	$0 \leq r \leq 31$	PC ← PC + 1	SP ← SP - 1

- POP
  - Loading stack contents back into a CPU register
    - POP is the opposite process of PUSH
    - SP is incremented
    - Top location of stack is copied back to the register

	Syntax:	Operands:	Program Counter:	Stack:
(i)	POP Rd	$0 \leq d \leq 31$	PC ← PC + 1	SP ← SP + 1

Initializing the Stack Pointer

- The SP register contains the value 0
  - When the AVR is powered on
- We must initialize the SP at the beginning of the program
  - Stack grows from higher memory Location to the lower memory location
  - It is common to initialize SP to uppermost memory location
    - Different AVRs have different amounts of RAM
    - In the AVR assembler, RAMEND represents the address of the last RAM location
    - To initialize SP
      - Load RAMEND into SP

CALL\RET instructions and the role of Stack

- When a subroutine is called
  - CPU saves the address of the instruction just bellow the CALL instruction on the stack
    - To know where to resume when it returns from called subroutine

PUSH (instruction bellow CALL) onto the stack

- When RET instruction at the end of the subroutine is executed
  - The top location of stack copied back to PC
  - SP is incremented

POP (instruction bellow CALL) into the PC

مثال

Toggle all the bits of Port B by sending to it the values $55 and $AA continuously. Put a time delay between each issuing of data to Port B.

```
.INCLUDE "M32DEF.INC"
.ORG 0
      LDI R16,HIGH(RAMEND)     ;load SPH
      OUT SPH,R16
      LDI R16,LOW(RAMEND)      ;load SPL
      OUT SPL,R16
```

Initialization of SP

```
BACK:
      LDI   R16,0x55      ;load R16 with 0x55
      OUT   PORTB,R16     ;send 55H to port B
      CALL  DELAY         ;time delay
      LDI   R16,0xAA      ;load R16 with 0xAA
      OUT   PORTB,R16     ;send 0xAA to port B
      CALL  DELAY         ;time delay
      RJMP  BACK          ;keep doing this indefinitely
```

Main program

```
;-------  this is the delay subroutine
      .ORG 0x300          ;put time delay at address 0x300
DELAY:
      LDI   R20,0xFF      ;R20 = 255,the counter
AGAIN:
      NOP                 ;no operation wastes clock cycles
      NOP
      DEC   R20
      BRNE  AGAIN         ;repeat until R20 becomes 0
      RET                 ;return to caller
```

Delay subroutine

Macros vs. Subroutine

Macros and subroutines are useful in writing assembly programs, but each has limitations. Macros increase code size every time they are invoked. For example, if you call a 10-instruction macro 10 times, the code size is increased by 100 instructions; whereas, if you call the same subroutine 10 times, the code size is only that of the subroutine instructions. On the other hand, a function call takes 3 or 4 clocks and the RET instruction takes 4 clocks to get executed. So, using functions adds around 8 clock cycles. The subroutines use stack space as well when called, while the macros do not.

AVR Timer Delay

- Delay subroutine
  - How to generate various time delays
  - How to calculate exact delays for AVR

- Two factors that can affect the accuracy of the delay
  - The crystal frequency
  - The AVR design

AVR Timer Delay

- Machine cycle
  - Certain amount of time for the CPU to execute an instruction
  - More instructions take no more than one or two machine cycles
  - The length of machine cycle depends on the frequency of the oscillator
  - One machine cycle consist of one oscillator period
    - To calculate the machine cycle
      - We take the inverse of the oscillator frequency

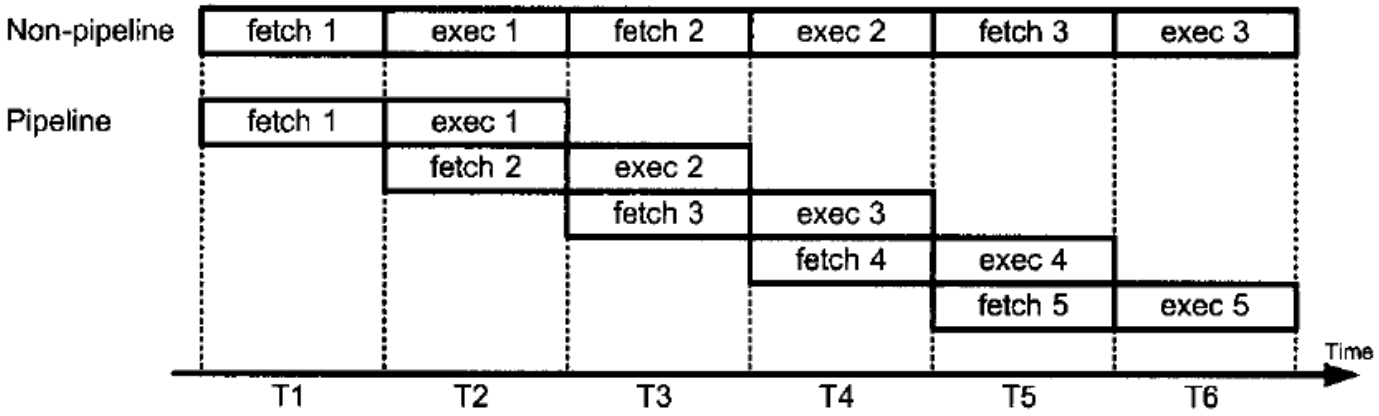
مثال

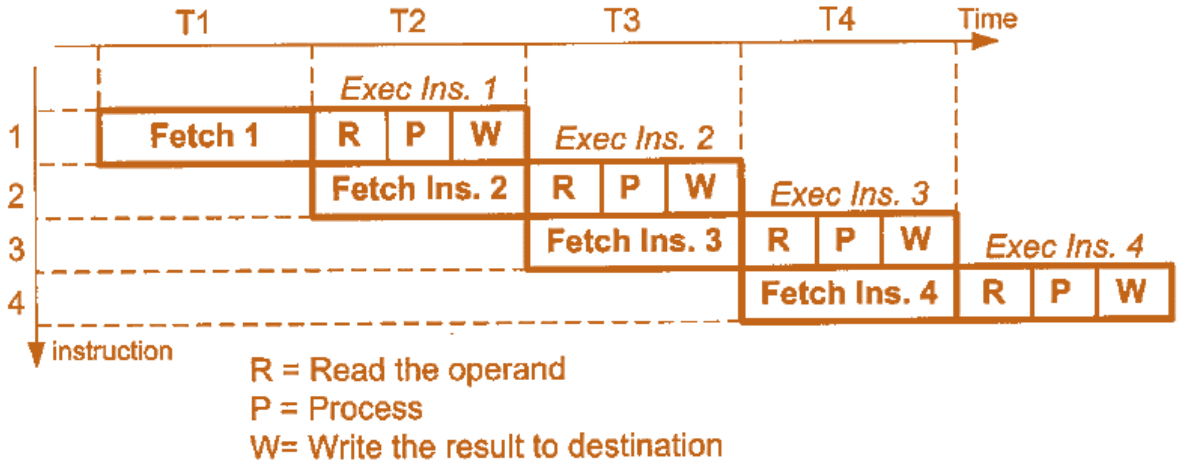
The following shows the crystal frequency for four different AVR-based systems. Find the period of the instruction cycle in each case.
(a) 8 MHz (b) 16 MHz (c) 10 MHz (d) 1 MHz

(a) instruction cycle is 1/8 MHz = 0.125 μs (microsecond) = 125 ns (nanosecond)
(b) instruction cycle = 1/16 MHz = 0.0625 μs = 62.5 ns (nanosecond)
(c) instruction cycle = 1/10 MHz = 0.1 μs = 100 ns
(d) instruction cycle = 1/1 MHz = 1 μs

Pipelining

- Overlapping of fetch and execution





Branch Penalty

- For the concept of pipelining to work
  - We need a buffer or queue
    - Which an instruction is prefetched and ready to be executed
- When a branch instruction is executed
  - CPU starts to fetch codes from new location
    - The code in the queue that was fetched previously is discarded
    - In this case, the execution unit must wait until the fetch unit fetches the new instruction
      - Called branch penalty

مثال

For an AVR system of 1 MHz, find how long it takes to execute each of the following instructions:

(a) LDI	(b) DEC	(c) LD
(d) ADD	(e) NOP	(f) JMP
(g) CALL	(h) BRNE	(i) .DEF

The machine cycle for a system of 1 MHz is 1 µs,

Instruction	*Instruction cycles*	*Time to execute*
(a) LDI	1	1 × 1 µs = 1 µs
(b) DEC	1	1 × 1 µs = 1 µs
(c) OUT	1	1 × 1 µs = 1 µs
(d) ADD	1	1 × 1 µs = 1 µs
(e) NOP	1	1 × 1 µs = 1 µs
(f) JMP	3	3 × 1 µs = 2 µs
(g) CALL	4	4 × 1 µs = 4 µs
(h) BRNE	2/1	(2 µs taken, 1 µs if it falls through)
(i) .DEF	0	(directive instructions do not produce machine instructions)

مثال

Find the size of the delay of the code snippet below if the crystal frequency is 10 MHz:

```
                                        Instruction Cycles
          .DEF COUNT = R20                      0
DELAY:    LDI      COUNT, 0xFF                  1
AGAIN:    NOP                                   1
          NOP                                   1
          DEC      COUNT                        1
          BRNE     AGAIN                        2/1
          RET                                   4
```

Therefore, we have a time delay of $[1 + ((1 + 1 + 1 + 2) \times 255) + 4] \times 0.1\ \mu s = 128.0\ \mu s$. Notice that BRNE takes two instruction cycles if it jumps back, and takes only one when falling through the loop. That means the above number should be 127.9 μs.

Delay Calculation for AVR

- Delay subroutine consists of two parts
  - Setting the counter
  - A loop
- Most of the time delay is performed by the body of the loop
- Very often
  - We calculate the time delay based in the instructions inside the loop
  - Ignore the clock cycles associated with the instructions outside of the loop

مثال

Find the size of the delay in the following program if the crystal frequency is 1 MHz:

```
.INCLUDE "M32DEF.INC"
.ORG 0
          LDI R16,HIGH(RAMEND) ;initialize SP
          OUT SPH,R16
          LDI R16,LOW(RAMEND)
          OUT SPL,R16
BACK:
          LDI R16,0x55         ;load R16 with 0x55
          OUT PORTB,R16        ;send 55H to port B
          RCALL DELAY          ;time delay
          LDI R16,0xAA         ;load R16 with 0xAA
          OUT PORTB,R16        ;send 0xAA to port B
          RCALL DELAY          ;time delay
          RJMP BACK            ;keep doing this indefinitely
;-----this is the delay subroutine
          .ORG 0x300           ;put time delay at address 0x300
DELAY:    LDI R20,0xFF         ;R20 = 255,the counter
AGAIN:
          NOP                  ;no operation wastes clock cycles
          NOP
          DEC R20
          BRNE AGAIN           ;repeat until R20 becomes 0
          RET                  ;return to caller
```

مثال

we have the following machine cycles for each instruction of the DELAY subroutine:

```
                                        Instruction Cycles
DELAY:   LDI  R20,0xFF                  1
AGAIN:   NOP                            1
         NOP                            1
         DEC  R20                       1
         BRNE AGAIN                     2/1
         RET                            4
```

Therefore, we have a time delay of $[1 + (255 \times 5) - 1 + 4] \times 1\ \mu s = 1279\ \mu s$.

پایان

موفق و پیروز باشید