PMR3303 – Eletrônica Digital para Mecatrônica
Prof. Dr. Celso M. Furukawa – Lista 15 – Entrega até **12/6/2019**

Os exercícios E1 a E3 se referem ao trecho de programa mostrado na figura f1, escrito em assembly do microcontrolador AVR ATMEGA. Nota: '$' indica constantes em hexadecimal.

E1. Descreva suscintamente o que o programa faz.

E2. Determine os endereços de cada instrução e determine os endereços associados aos rótulos (*labels*) Main, Loop e Ahead.

E3. Mostre como o assembler codificará em binário as seguintes instruções: "jmp Main", "breq Ahead" (primeira ocorrência) e "rjmp Loop".

```
.org 0
        jmp Main
Main:   clr r16
        ldi r30, $00
        ldi r31, $01
        ldi r26, $10
        ldi r27, $01
Loop:   ld r0, z+
        st x+, r0
        or r0, r0
        sts $0300, r16
        breq Ahead
        cpi r16, $0f
        breq Ahead
        inc r16
        rjmp Loop
Ahead:  break
```

Figura f1

```
se (R0 ≠ R1) então
    R0 ← R1
    R1 ← R1 + 1
senão
    R1 ← 5
R2 ← R0 + R1
```

Figura f4

```
X ← 0x0101
R0 ← 0
R1 ← SRAM[X]
while (R1 != 0)
   R0 ← R0 + 1
   X ← X + 1
   R1 ← SRAM[X]
SRAM[0x0100] ← R0
```

Figura f5

E4. Codifique um programa em assembly do AVR que implemente o algorítmo descrito na figura f4.

E5. Repita para o algorítmo da figura f5, onde X representa o par de registradores R27:R26.

Para fazer os exercícios E6 a E8, considere que a tabela t6 mostra algumas instruções de um controlador hipotético de 12 bits: instruções, dados, palavras de memória e registradores possuem 12 bits. A arquitetura é de Von Neumann (memória de programa e de dados ocupam o mesmo espaço de endereçamento). Na tabela, *d*, *k* e *r* representam um digito hexadecimal (de 4 bits); R0 a R15 são registradores de uso geral (R15 também pode ser usado como registrador de índice X); PC é *Program Counter*, e Z é um *flag* que indica resultado nulo quando em 1. Mem[*kkk*] indica a posição de memória de endereço *kkk*. Na coluna *Words* tem-se o comprimento da instrução em número de palavras (de 12 bits). A figura f6 mostra um trecho de programa em *assembly* desse processador.

E6. Transcreva o trecho de programa para o código de máquina do processador, alocando-o a partir da posição 0x010 (em hexadecimal) de memória. Indique constantes e endereços em hexadecimal.

E7. Descreva em poucas palavras o que esse trecho de programa faz.

E8. Suponha que PC seja iniciado com o endereço 0x012. Transcreva em *assembly* o programa que seria executado.

Para fazer os exercícios E9 a E10, considere que a tabela t9 mostra algumas instruções do controlador M68HCS08 de 8 bits. Mem[*i*] indica a posição *i* de memória; A e X são registradores de 8 bits, sendo A o acumulador e X o registrador de índice; PC é *Program Counter*; Z é um *flag* que indica resultado nulo quando em 1; *opr* representa um operando (constante) de 8 bits. Na coluna *Bytes* tem-se o comprimento da instrução em número de bytes. A figura f9 mostra um trecho de programa em *assembly*, onde '#15h' representa o valor imediato (armazenado juntamente com o programa) 15h (em hexa). A diretiva *DB* aloca um byte, de modo que *sum* e *inx* representam variáveis de 8 bits alocadas na memória nos endereços 80h e 81h (devido a diretiva '*org 80h*').

E9. Transcreva o trecho de programa para o código de máquina do processador, alocando-o a partir da posição 0h de memória. Indique constantes e endereços em hexadecimal.

E10. Assumindo que *sum* e *inx* começam zerados, descreva em poucas palavras o que esse trecho de programa faz.

Tabela t6

Mneum.	Hexa	Descrição	Words
ADD R*d*, R*r*	4*dr*	R*d* ← R*d* + R*r*	1
CLR R*d*	30*d*	R*d* ← 0	1
CPI R*d*, *kkk*	10*d*	Faz (R*d* – *kkk*) e ajusta flag Z	2
INC R*d*	80*d*	R*d* ← R*d* + 1	1
JMP *kkk*	C00	PC ← *kkk*	2
JPEQ *kkk*	D00	se Z = 1, PC ← *kkk*	2
LD R*d*, X	00*d*	R*d* ← Mem[R15]	1
LDI R*d*, *kkk*	E0*d*	R*d* ← *kkk*	2
LDS R*d*, *kkk*	01*d*	R*d* ← Mem[*kkk*]	2
MOV R*d*, R*r*	2*dr*	R*d* ← R*r*	1
NOP	FFF	PC ← PC + 1	1
ST X, R*r*	A0*r*	Mem[R15] ← R*r*	1
STS *kkk*, R*r*	A1*r*	Mem[*kkk*] ← R*r*	2

```
        .ORG 0x010
        CLR R0
        LDI R15, 0x010
Loop:   CPI R15, 0x01F
        JPEQ Out
        LD R3, X
        ADD R0, R3
        INC R15
        JMP Loop
Out:    STS 0x100, R0
        NOP
```

Figura f6

Tabela t4

Mneum.	Hexa	Descrição	Bytes
ADD ,X	FB	A ← A + Mem[X]	1
BEQ *opr*	27	se Z = 1, PC ← PC + 2 + *opr*	2
BRA *opr*	20	PC ← PC + 2 + *opr*	2
CMP #*opr*	A1	Faz (A – *opr*) e ajusta flag Z	2
INCA	4C	A ← A + 1	1
LDA *opr*	B6	A ← Mem[*opr*]	2
NOP	9D	PC ← PC + 1	1
STA *opr*	B7	Mem[*opr*] ← A	2
TAX	97	X ← A	1

```
        org 0h
        LDA inx
Loop:   CMP #15h
        BEQ Out
        TAX
        LDA sum
        ADD ,X
        STA sum
        LDA inx
        INCA
        STA inx
        BRA Loop
Out:    NOP

        org 80h
sum     DB
inx     DB
```

Figura f4

E2) Main: $02; Loop: $07; Ahead: $11.
E3) "jmp Main": 940C 0002; "breq Ahead": F021; "rjmp Loop": CFF6
E6) (Endereço: instrução) 10: 300 / 11: E0F 010 / 13: 10F 01F / 15: D00 01C / 17: 003 / 18: 403 / 19: 80F / 1A: C00 013 / 1C: A10 100 / 1E: FFF.
E7) Soma as instruções do próprio programa em R0 e grava o resultado no endereço 0x100.
E8) LDS R0, 0x10F / LDS R15, 0xD00 / LDS R12, 0x003 / ADD R0, R3 / INC R15 / etc (programa sem sentido).
E9) (Endereço: instrução) 00: B6 81 / 02: A1 15 / 04: 27 0D / 06: 97 / 07: B6 80 / 09: FB / 0A: B7 80 / 0C: B6 81 / 0E: 4C / 0F: B7 81 / 11: 20 EF (-17 em C2) / 13: 9D.
E10) Soma as instruções do próprio programa.