BC-1503

Arquitetura de Computadores

Instruções e Linguagem de Máquina

Guiou Kobayashi

guiou.kobayashi@ufabc.edu.br

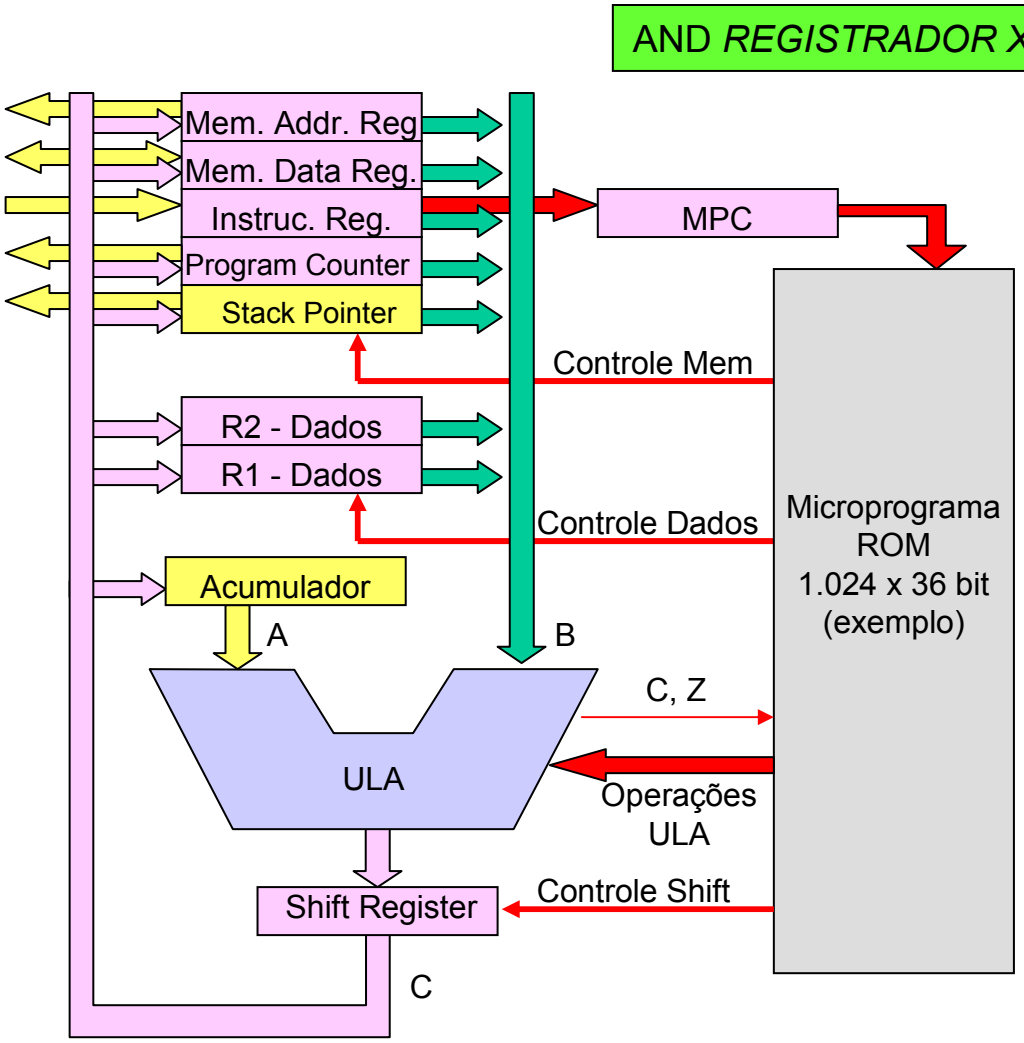
2º Quadrimestre, 2014

CONTEÚDO PROGRAMÁTICO:

- História e Evolução dos Computadores e Sistemas
- Estrutura de Computadores Digitais
- Lógica Digital Binária
- Processamento
- **Instruções e linguagem de máquina**
- Microprocessadores modernos: pipeline, super escalar, RISC
- Memórias cache e gerenciamento de memórias
- Arquitetura de computadores pessoais
- Arquitetura de Computadores Paralelos
- Sistemas Computacionais: desempenho e confiabilidade

Tipos de Instruções

INSTRUÇÕES DE UM BYTE



Exemplo:

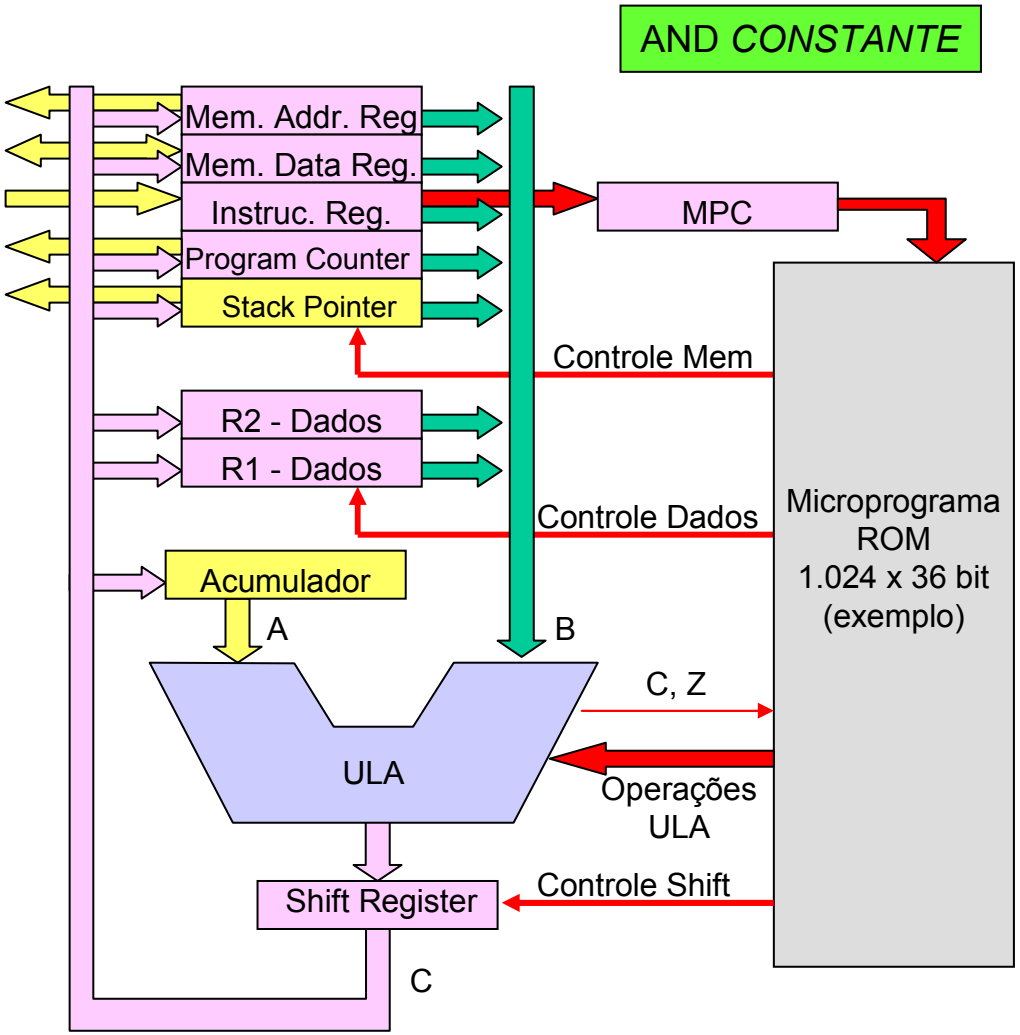
AND ACC, R1: **(ACC ← ACC *AND* R1)**

1: FETCH INSTRUÇÃO
Q1: PC → Via de Endereços
Q2: Comando de Leitura
Q3: Memória → Via de Dados
Q4: Via de Dados → IR

2: Acumulador = Acumulador AND R1
Q1: R1 → Barramento B
Q2: ULA: Operação AND
Q3: ULA: Retenção Barramento A
Q4: Barramento C → Acumulador

3: PC = PC + 1
Q1: ULA: Bloqueia Barramento A
Q2: PC → Barramento B
Q3: ULA: Incrementa Barramento B
Q4: Barramento C → PC

INSTRUÇÕES DE DOIS BYTES



Exemplo:

AND ACC, 5A: **(ACC ← ACC *AND* 5A)**

1: FETCH INSTRUÇÃO

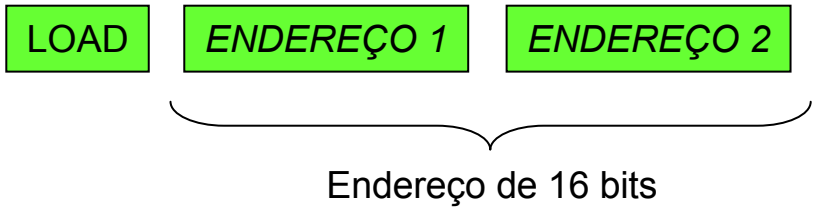
2: PC = PC + 1

3: FETCH CONSTANTE

4: Acum. = Acum. AND Constante

5: PC = PC + 1

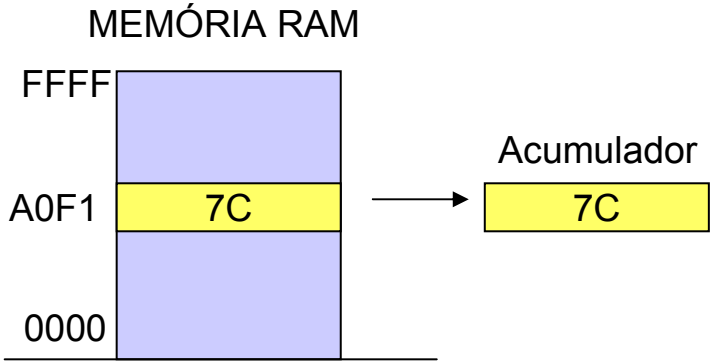
INSTRUÇÕES DE TRÊS BYTES - 1



LEITURA DE UM DADO DA MEMÓRIA RAM: INSTRUÇÃO LOAD

Especificar o endereço de 16 bits da posição da memória a ser lida

Exemplo: leitura da posição A0F1, que contém o valor 7C



Exemplo:

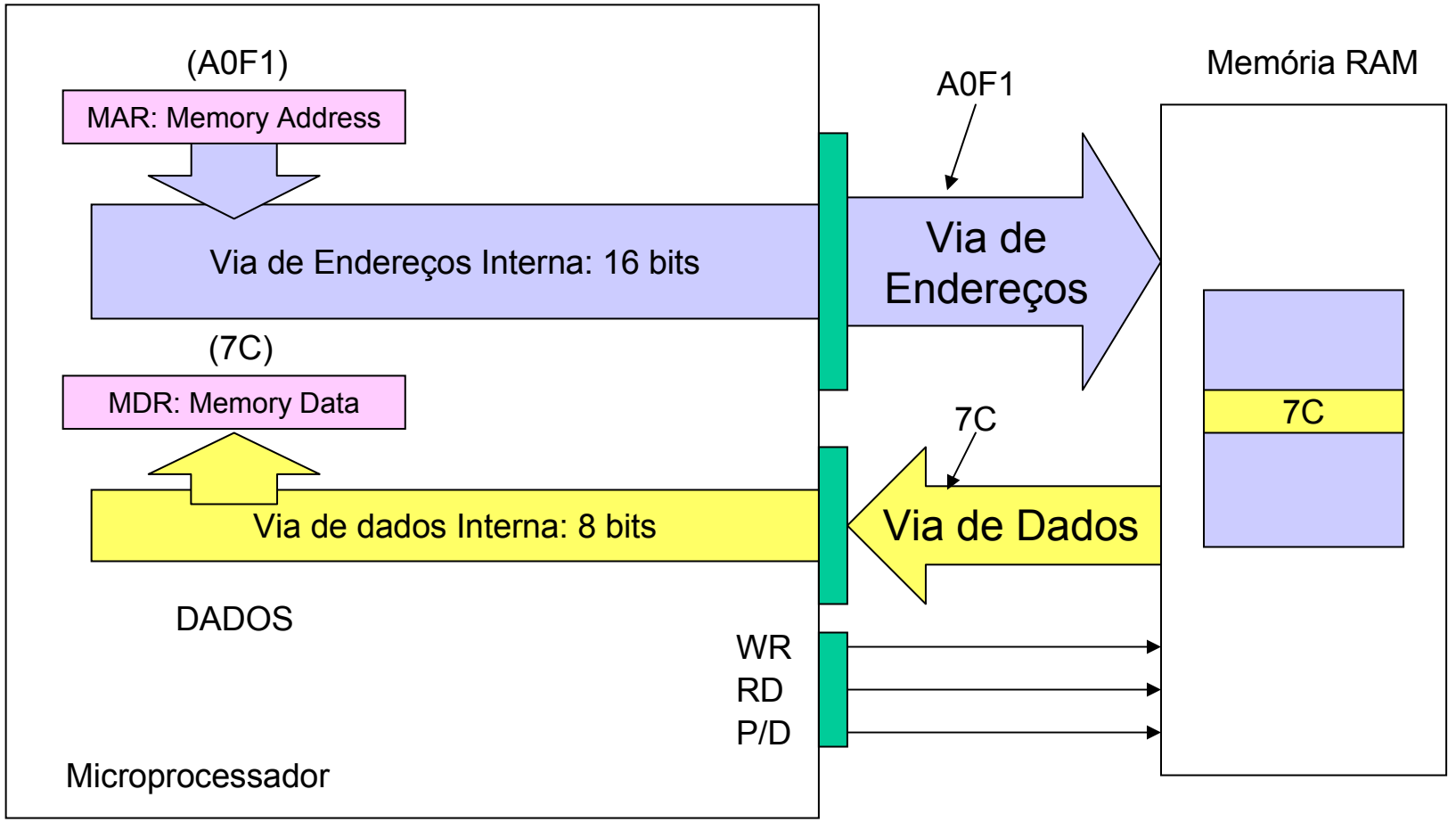
LOAD A0F1: (ACC ← [A0F1])

1: FETCH INSTRUÇÃO
2: PC = PC + 1
3: FETCH ENDEREÇO 1 (+signif.)
4: PC = PC + 1
5: FETCH ENDEREÇO 2 (-signif.)
6: MDR ← [ENDEREÇO]
Q1: MAR → Via de Endereços
Q2: Comando de Leitura
Q3: Memória → Via de Dados
Q4: Via de Dados → MDR

7: Transferência para Acumulador
ACC ← MDR
Q1: ULA: Bloqueia Barramento A
Q2: MDR → Barramento B
Q3: ULA: bypass Barramento B
Q4: Barramento C → Acumulador

8: PC = PC + 1

FLUXO DE INFORMAÇÕES PARA LEITURA



INSTRUÇÕES DE TRÊS BYTES - 2

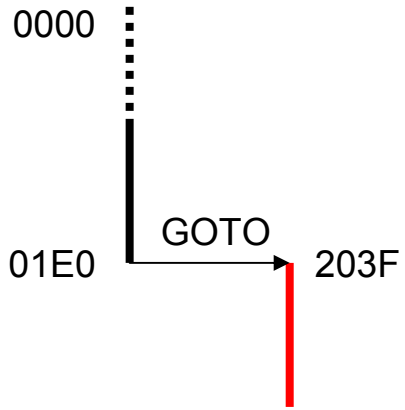
SALTO INCONDICIONAL: INSTRUÇÃO GOTO ou JUMP

Especificar o endereço de 16 bits da próxima posição do programa

Exemplo:

GOTO 203F

1: FETCH INSTRUÇÃO
2: PC = PC + 1
3: FETCH ENDEREÇO 1 (+signif.)
4: PC = PC + 1
5: FETCH ENDEREÇO 2 (-signif.)
6: PC ← ENDEREÇO

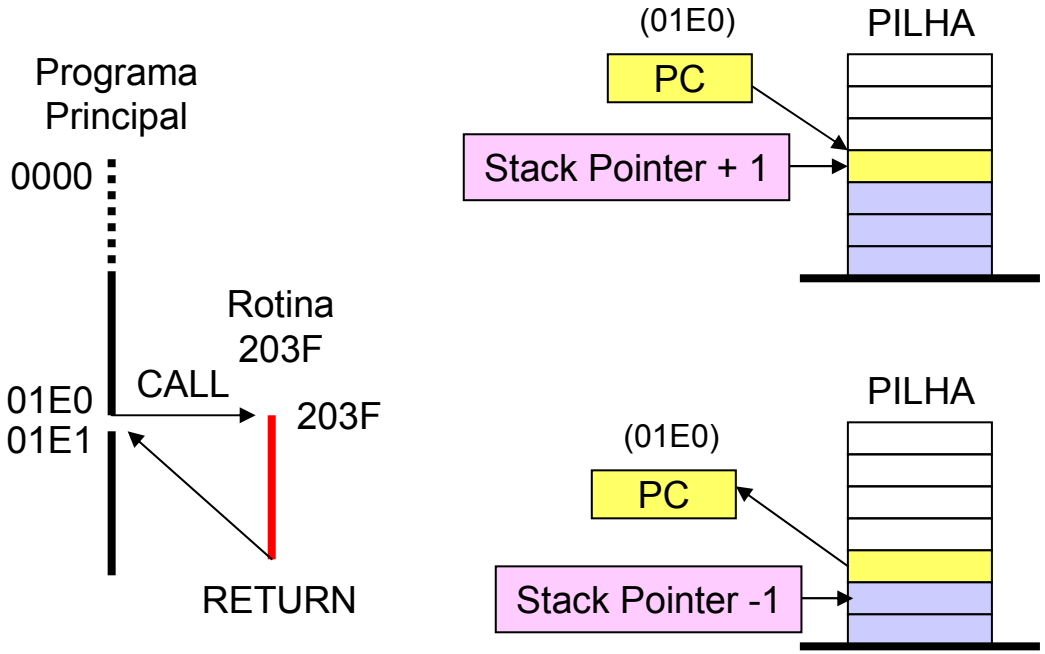


	Program Counter
Antes da instrução GOTO:	01E0
Depois da instrução:	203F

INSTRUÇÕES DE TRÊS BYTES - 3

CHAMADA DE ROTINA: INSTRUÇÃO CALL

Especificar o endereço de 16 bits da posição da rotina



Exemplo:

CALL 203F

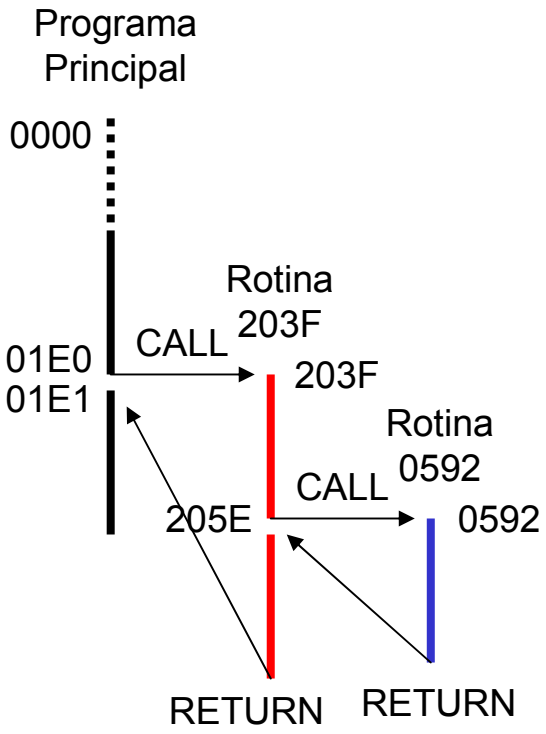
1: FETCH INSTRUÇÃO
2: PC = PC + 1
3: FETCH ENDEREÇO 1 (+signif.)
4: PC = PC + 1
5: FETCH ENDEREÇO 2 (-signif.)
6: SP = SP + 1
7: [SP] ← PC
8: PC ← ENDEREÇO

RETURN

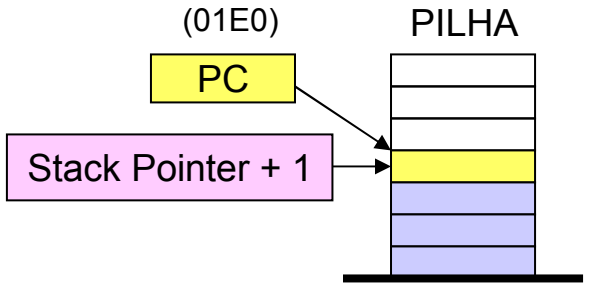
1: FETCH INSTRUÇÃO
2: PC ← [SP]
3: SP = SP - 1
4: PC = PC + 1

NÍVEIS DE SUB-ROTINAS

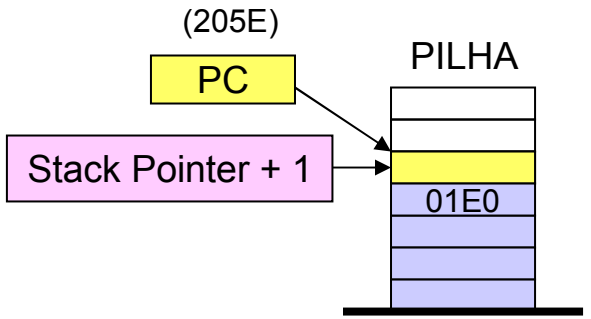
Chamada de Rotinas de dentro das Rotinas



CALL 203F



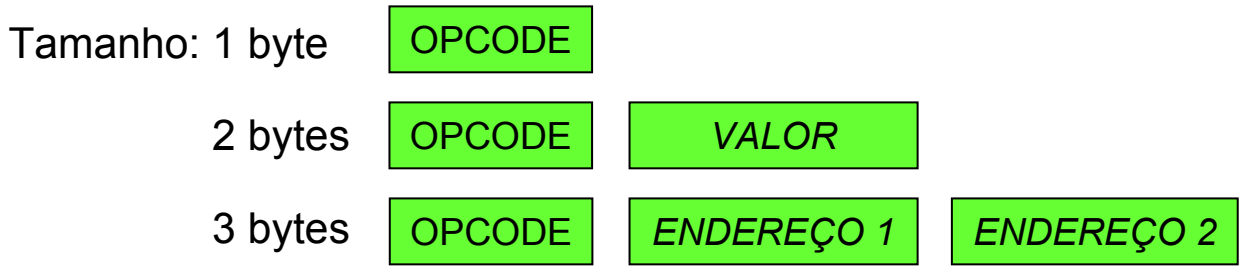
CALL 0592



Estouro da Pilha: Stack Pointer Overflow

INSTRUÇÕES: RESUMO

- Instruções são executadas em ciclos de máquinas, que estabelecem os passos necessários até a sua execução. O número de ciclos depende da instrução.
- Instruções possuem tamanhos diferentes em bytes, sendo compostas por um código de operação (Op Codes) seguido por parâmetros que podem ser valores constantes, endereços, etc.



- Etapas típicas de um ciclo de instruções:
  1 - Fetch da instrução, da memória para o registrador de instruções (IR)
  2 - Incrementa o contador de instruções (PC), apontando-o para a próxima instrução
  3 - Determina o tipo de instrução carregada (Decode)
  4 - Se a instrução utiliza mais parâmetros, busca por fetch sucessivos
  5 - Se a instrução realiza acesso à memória (leitura ou escrita), efetua o acesso
  6 - Executa a instrução
  7 - Vai para a etapa 1 para iniciar a execução da próxima instrução

Exemplos de Linguagens de Máquina

EXEMPLO DE ASSEMBLY DE UM PROCESSADOR HIPOTÉTICO

OBS: PROCESSADOR DE 8 BITs E ENDEREÇAMENTO DE 16 BITs

OPERAÇÕES COM INTEIROS

ADD	1-2 (1 ou 2 bytes)	
SUB	1-2	
AND	1-2	*OPERANDO*:
IOR	1-2	- Registrador (1 byte)
XOR	1-2	- Constante (2 bytes)
COMP	1	
CLR	1	*FORMATO*:
INC	1	Instrução *Destino*, *Origem*
DEC	1	

OPERAÇÕES DE DESLOCAMENTO

RL	1 (Rotate Left Acumulador)
RR	1 (Rotate Right Acumulador)

OPERAÇÕES DE FLUXO DE DADOS

MOV	1 (Acumulador ← → Registrador)
MOVC	1 (Acumulador ← Constante)
PUSH	1 (Insere na Pilha)
POP	1 (Retira da Pilha)
LOAD	3 (Memória: Endereço 2 bytes)
SAVE	3 (Memória: Endereço 2 bytes)
IN	3 (Dispositivos I/O: Endereço 2 bytes)
OUT	3 (Dispositivos I/O: Endereço 2 bytes)

BITs DE ESTADO (do Acumulador)

Z	(Zero): = 1 se Acumulador = 0
C	(Carry): = se Vai-um após soma

REGISTRADORES ESPECIAIS

ACC	(Acumulador)
PC	(Program Counter)
IR	(Instruction Register)
SP	(Stack Pointer)
MAR	(Memory Address Reg)
MDR	(Memory Data Reg)
IND	(Indirect Address Reg)

REGISTRADORES DE DADOS

R0 a R7 Registradores de uso geral

OPERAÇÕES DE FLUXO DE CONTROLE

GOTO	3 (Endereço 2 bytes)
CALL	3
RET	1 (Return)
	Condicionais
JZ	3 (Jump on Zero)
JNZ	3 (Jump on Non-Zero)
JC	3 (Jump on Carry)
JNC	3 (Jump in Non-Carry)

NOP 1 (No Operation)

INSTRUÇÕES DE MÁQUINA: Exemplos Reais

Loads

LDSB ADDR,DST	Load signed byte (8 bits)
LDUB ADDR,DST	Load unsigned byte (8 bits)
LDSH ADDR,DST	Load signed halfword (16 bits)
LDUH ADDR,DST	Load unsigned halfword (16)
LDSW ADDR,DST	Load signed word (32 bits)
LDUW ADDR,DST	Load unsigned word (32 bits)
LDX ADDR,DST	Load extended (64-bits)

Stores

STB SRC,ADDR	Store byte (8 bits)
STH SRC,ADDR	Store halfword (16 bits)
STW SRC,ADDR	Store word (32 bits)
STX SRC,ADDR	Store extended (64 bits)

Arithmetic

ADD R1,S2,DST	Add
ADDCC "	Add and set icc
ADDC "	Add with carry
ADDCCC "	Add with carry and set icc
SUB R1,S2,DST	Subtract
SUBCC "	Subtract and set icc
SUBC "	Subtract with carry
SUBCCC "	Subtract with carry and set icc
MULX R1,S2,DST	Multiply
SDIVX R1,S2,DST	Signed divide
UDIVX R1,S2,DST	Unsigned divide
TADCC R1,S2,DST	Tagged add

Shifts/rotates

SLL R1,S2,DST	Shift left logical (32 bits)
SLLX R1,S2,DST	Shift left logical extended (64)
SRL R1,S2,DST	Shift right logical (32 bits)
SRLX R1,S2,DST	Shift right logical extended (64)
SRA R1,S2,DST	Shift right arithmetic (32 bits)
SRAX R1,S2,DST	Shift right arithmetic ext. (64)

Boolean

AND R1,S2,DST	Boolean AND
ANDCC "	Boolean AND and set icc
ANDN "	Boolean NAND
ANDNCC "	Boolean NAND and set icc
OR R1,S2,DST	Boolean OR
ORCC "	Boolean OR and set icc
ORN "	Boolean NOR
ORNCC "	Boolean NOR and set icc
XOR R1,S2,DST	Boolean XOR
XORCC "	Boolean XOR and set icc
XNOR "	Boolean EXCLUSIVE NOR
XNORCC "	Boolean EXCL. NOR and set icc

Transfer of control

BPcc ADDR	Branch with prediction
BPr SRC,ADDR	Branch on register
CALL ADDR	Call procedure
RETURN ADDR	Return from procedure
JMPL ADDR,DST	Jump and Link
SAVE R1,S2,DST	Advance register windows
RESTORE "	Restore register windows
Tcc CC,TRAP#	Trap on condition
PREFETCH FCN	Prefetch data from memory
LDSTUB ADDR,R	Atomic load/store
MEMBAR MASK	Memory barrier

Miscellaneous

SETHI CON,DST	Set bits 10 to 31
MOVcc CC,S2,DST	Move on condition
MOVr R1,S2,DST	Move on register
NOP	No operation
POPC S1,DST	Population count
RDCCR V,DST	Read condition code register
WRCCR R1,S2,V	Write condition code register
RDPC V,DST	Read program counter

SRC = source register
DST = destination register
R1 = source register
S2 = source: register or immediate
ADDR = memory address

TRAP# = trap number
FCN = function code
MASK = operation type
CON = constant
V = register designator

CC = condition code set
R =destination register

cc = condition
r = LZ,LEZ,Z,NZ,GZ,GEZ

Figure 5-34. The primary UltraSPARC II integer instructions.

Loads

typeLOAD IND8	Push local variable onto stack
typeALOAD	Push array element on stack
BALOAD	Push byte from an array on stack
SALOAD	Push short from an array on stack
CALOAD	Push char from an array on stack
AALOAD	Push pointer from an array on "

Stores

typeSTORE IND8	Pop value and store in local var
typeASTORE	Pop value and store in array
BASTORE	Pop byte and store in array
SASTORE	Pop short and store in array
CASTORE	Pop char and store in array
AASTORE	Pop pointer and store in array

Pushes

BIPUSH CON8	Push a small constant on stack
SIPUSH CON16	Push 16-bit constant on stack
LDC IND8	Push constant from const pool
typeCONST_#	Push immediate constant
ACONST_NULL	Push a null pointer on stack

Arithmetic

typeADD	Add
typeSUB	Subtract
typeMUL	Multiple
typeDIV	Divide
typeREM	Remainder
typeNEG	Negate

Boolean/shift

iIAND	Boolean AND
iIOR	Boolean OR
iIXOR	Boolean EXCLUSIVE OR
iISHL	Shift left
iISHR	Shift right
iIUSHR	Unsigned shift right

Conversion

x2y	Convert x to y
i2c	Convert integer to char
i2b	Convert integer to byte

Stack management

DUPxx	Six instructions for duping
POP	Pop an int from stk and discard
POP2	Pop two ints from stk and discard
SWAP	Swap top two ints on stack

Comparison

IF_ICMPrel OFFSET16	Conditional branch
IF_ACMPEQ OFFSET16	Branch if two ptrs equal
IF_ACMPNE OFFSET16	Branch if ptrs unequal
IFrel OFFSET16	Test 1 value and branch
IFNULL OFFSET16	Branch if ptr is null
IFNONNULL OFFSET16	Branch if ptr is nonnull
LCMP	Compare two longs
FCMPL	Compare 2 floats for <
FCMPG	Compare 2 floats for >
DCMPL	Compare doubles for <
DCMPG	Compare doubles for >

Transfer of control

INVOKEVIRTUAL IND16	Method invocation
INVOKESTATIC IND16	Method invocation
INVOKEINTERFACE ...	Method invocation
INVOKESPECIAL IND16	Method invocation
JSR OFFSET16	Invoke finally clause
typeRETURN	Return value
ARETURN	Return pointer
RETURN	Return void
RET IND8	Return from finally
GOTO OFFSET16	Unconditional branch

Arrays

ANEWARRAY IND16	Create array of ptrs
NEWARRAY ATYPE	Create array of atype
MULTINEWARRAY IN16,D	Create multidim array
ARRAYLENGTH	Get array length

Miscellaneous

IINC IND8,CON8	Increment local variable
WIDE	Wide prefix
NOP	No operation
GETFIELD IND16	Read field from object
PUTFIELD IND16	Write field to object
GETSTATIC IND16	Get static field from class
NEW IND16	Create a new object
INSTANCEOF OFFSET16	Determine type of obj
CHECKCAST IND16	Check object type
ATHROW	Throw exception
LOOKUPSWITCH ...	Sparse multiway branch
TABLESWITCH ...	Dense multiway branch
MONITORENTER	Enter a monitor
MONITOREXIT	Leave a monitor

IND8/16 = index of local variable
CON8/16, D, ATYPE = constant

type, x, y = I, L, F, D
OFFSET16 for branch

The JVM instruction set.

Moves

MOV DST,SRC	Move SRC to DST
PUSH SRC	Push SRC onto the stack
POP DST	Pop a word from the stack to DST
XCHG DS1,DS2	Exchange DS1 and DS2
LEA DST,SRC	Load effective addr of SRC into DST
CMOV DST,SRC	Conditional move

Arithmetic

ADD DST,SRC	Add SRC to DST
SUB DST,SRC	Subtract DST from SRC
MUL SRC	Multiply EAX by SRC (unsigned)
IMUL SRC	Multiply EAX by SRC (signed)
DIV SRC	Divide EDX:EAX by SRC (unsigned)
IDIV SRC	Divide EDX:EAX by SRC (signed)
ADC DST,SRC	Add SRC to DST, then add carry bit
SBB DST,SRC	Subtract DST & carry from SRC
INC DST	Add 1 to DST
DEC DST	Subtract 1 from DST
NEG DST	Negate DST (subtract it from 0)

Binary coded decimal

DAA	Decimal adjust
DAS	Decimal adjust for subtraction
AAA	ASCII adjust for addition
AAS	ASCII adjust for subtraction
AAM	ASCII adjust for multiplication
AAD	ASCII adjust for division

Boolean

AND DST,SRC	Boolean AND SRC into DST
OR DST,SRC	Boolean OR SRC into DST
XOR DST,SRC	Boolean Exclusive OR SRC to DST
NOT DST	Replace DST with 1's complement

Shift/rotate

SAL/SAR DST,#	Shift DST left/right # bits
SHL/SHR DST,#	Logical shift DST left/right # bits
ROL/ROR DST,#	Rotate DST left/right # bits
RCL/RCR DST,#	Rotate DST through carry # bits

Test/compare

TST SRC1,SRC2	Boolean AND operands, set flags
CMP SRC1,SRC2	Set flags based on SRC1 - SRC2

Transfer of control

JMP ADDR	Jump to ADDR
Jxx ADDR	Conditional jumps based on flags
CALL ADDR	Call procedure at ADDR
RET	Return from procedure
IRET	Return from interrupt
LOOPxx	Loop until condition met
INT ADDR	Initiate a software interrupt
INTO	Interrupt if overflow bit is set

Strings

LODS	Load string
STOS	Store string
MOVS	Move string
CMPS	Compare two strings
SCAS	Scan Strings

Condition codes

STC	Set carry bit in EFLAGS register
CLC	Clear carry bit in EFLAGS register
CMC	Complement carry bit in EFLAGS
STD	Set direction bit in EFLAGS register
CLD	Clear direction bit in EFLAGS reg
STI	Set interrupt bit in EFLAGS register
CLI	Clear interrupt bit in EFLAGS reg
PUSHFD	Push EFLAGS register onto stack
POPFD	Pop EFLAGS register from stack
LAHF	Load AH from EFLAGS register
SAHF	Store AH in EFLAGS register

Miscellaneous

SWAP DST	Change endianness of DST
CWQ	Extend EAX to EDX:EAX for division
CWDE	Extend 16-bit number in AX to EAX
ENTER SIZE,LV	Create stack frame with SIZE bytes
LEAVE	Undo stack frame built by ENTER
NOP	No operation
HLT	Halt
IN AL,PORT	Input a byte from PORT to AL
OUT PORT,AL	Output a byte from AL to PORT
WAIT	Wait for an interrupt

SRC = source
DST = destination

# = shift/rotate count
LV = # locals

Figure 5-33. A selection of the Pentium II integer instructions.

Exercício

EXERCÍCIO DE PROGRAMAÇÃO EM ASSEMBLY

Utilizando as instruções da página anterior, desenvolver em linguagem de máquina (Assembly) um programa que implementa um relógio digital.

Dados:
- um gerador de sinais no endereço F000, que gera pulsos a cada 1 segundo;
- um display no endereço F001, que apresenta as horas;
- um display no endereço F002, que apresenta os minutos;
- um display no endereço F003, que apresenta os segundos.

Utilizar rótulos (labels) para identificar posições de endereços no programa.
O gerador de sinais e displays são dispositivos de E/S (I/O)