
Elementos de Lógica Digital II

COMP09 μ Processor

Vanderlei Bonato

Eduardo Simões

Introduction

- **Architecture:** the programmer's view of the computer
 - Defined by instructions (operations) and operand locations
- **Microarchitecture:** how to implement an architecture in hardware

Application Software	programs
Operating Systems	device drivers
Architecture	instructions registers
Micro-architecture	datapaths controllers
Logic	adders memories
Digital Circuits	AND gates NOT gates
Analog Circuits	amplifiers filters
Devices	transistors diodes
Physics	electrons

Assembly Language

- To command a computer, you must understand its language.
 - **Instructions:** words in a computer's language
 - **Instruction set:** the vocabulary of a computer's language
- Instructions indicate the operation to perform and the operands to use.
 - **Assembly language:** human-readable format of instructions
 - **Machine language:** computer-readable format
 - Computers only understand 1's and 0's
 - Binary representation of instructions

Stored Program

- Sequence of instructions: only difference between two applications (for example, a text editor and a video game)
- To run a new program:
 - No rewiring required
 - Simply store new program in memory
- The processor hardware executes the program:
 - *fetches* (reads) the instructions from memory in sequence
 - performs the specified operation
- The program counter (PC) keeps track of the current instruction

What needs to be stored in memory?

- Instructions (also called *text*)
- Data
 - Global/static: allocated before program begins
 - Dynamic: allocated within program
- How big is memory?
 - At most $2^{32} = 4$ gigabytes (4 GB)
 - From address 0x00000000 to 0xFFFFFFFF

Ada Lovelace, 1815 - 1852

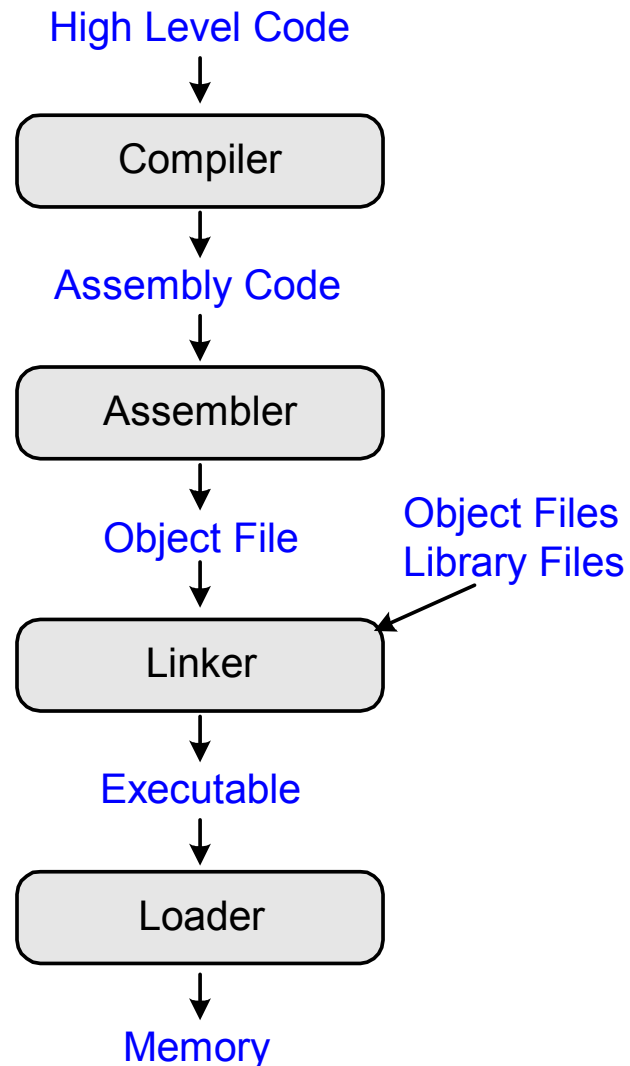
- Wrote the first computer program
- Her program calculated the Bernoulli numbers on Charles Babbage's Analytical Engine
- She was the only legitimate child of the poet Lord Byron



Programming

- High-level languages:
 - e.g., C, Java, Python
 - Written at more abstract level
- Common high-level software constructs:
 - if/else statements
 - for loops
 - while loops
 - array accesses
 - procedure calls
- Other useful instructions:
 - Arithmetic/logical instructions
 - Branching

How do we compile & run an application?



Grace Hopper, 1906 - 1992

- Graduated from Yale University with PhD in mathematics
- Developed first compiler
- Helped develop the COBOL programming language
- Highly awarded naval officer
- Received World War II Victory Metal and National Defense Service Medal, among others



RISC x CISC architecture

- RISC (Reduced Instruction Set Computer)
 - A small number of simple instructions
 - Hardware to decode and execute the instruction can be simple, small, and fast.
 - More complex instructions (that are less common) can be performed using multiple simple instructions.
- Other architectures, such as Intel's IA-32 found in many PC's, are **complex instruction set computers (CISC)**. They include complex instructions that are rarely used, such as the "string move" instruction that copies a string (a series of characters) from one part of memory to another.

The COMP09 μ Processor

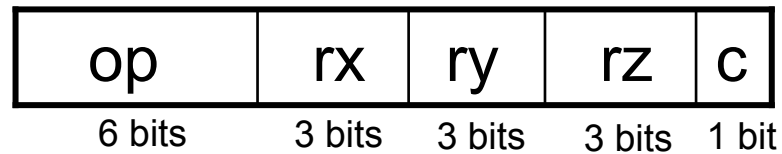
- COMP09 architecture:
 - Developed by Eduardo do Valle Simões and Vanderlei Bonato at ICMC/USP.
 - RISC architecture type
- 16-bit instructions
- Four instruction formats:
 - R-type: register operands
 - JB-type: Jump and Branch instruction type
 - C-type: system control
 - S-type: stack operations

The uCOMP09 register set

Name	Qtt	Purpose
R_n	0-7	General purpose
FR	1	Bits of flag
SP	1	Stack pointer
PC	1	Program counter
IR (interno)	1	Instruction register
MAR (interno)	1	Memory address buffer
MBR (interno)	1	Data memory buffer

Instruction format

- Register operands (R-type)
 - op = opcode
 - rx, ry, rz: registers
 - c: bit of carry



Data manipulation instructions

Direct address mode

STORE END RX	MEM(END) <- RX	110001 RX xxx xxx x END
LOAD RX END	RX <- MEM(END)	110000 RX xxx xxx x END

Indirect address mode

STORE RX NR	MEM(RX) <- #NR	111001 RX xxx xxx x NR
LOAD RX NR	RX <- #NR	111000 RX xxx xxx x NR

Indexed address mode

STORE RY RX	MEM(RY) <- RX	111101 RX RY xxx x
LOAD RX RY	RX <- MEM(RY)	111100 RX RY xxx x
MOV RX RY	RX <- RY	110011 RX RY xxx x

Arithmetic instructions

ADD RZ RX RY	$RZ \leftarrow RX + RY$	100000 RX RY RZ 0
ADDC RZ RX RY	$RZ \leftarrow RX + RY + C$	100000 RX RY RZ 1
SUB RZ RX RY	$RZ \leftarrow RX - RY$	100001 RX RY RZ 0
SUBC RZ RX RY	$RZ \leftarrow RX - RY + C$	100001 RX RY RZ 1
MULT RZ RX RY	$RZ \leftarrow RX * RY$	100010 RX RY RZ 0
MULTC RZ RX RY	$RZ \leftarrow RX * RY + C$	100010 RX RY RZ 1
DIV RZ RX RY	$RZ \leftarrow RX / RY$	100011 RX RY RZ 0
DIVC RZ RX RY	$RZ \leftarrow RX / RY + C$	100011 RX RY RZ 1
INC RX	$RX++$	100100 RX xxx xxx 0
DEC RX	$RX--$	100101 RX xxx xxx 1
COMP RX RY	$FR \leftarrow \text{COND}$	000101 RX RY xxx x

Logic instructions

AND RZ RX RY	RZ<-RX AND RY	010010 RX RY RZ x
OR RZ RX RY	RZ<-RX OR RY	010011 RX RY RZ x
XOR RZ RX RY	RZ<-RX XOR RY	010100 RX RY RZ x
NOT RY RX	RY<-NOT(RX)	010101 RX RY xxx x
ROT L RX	ROTATE TO LEFT	010000 RX xxx x1x 0
ROT R RX	ROTATE TO RIGHT	010000 RX xxx x1x 1
SHIFT L0 RX	SHIFT TO LEFT (FILL 0)	010000 RX xxx x00 0
SHIFT L1 RX	SHIFT TO LEFT (FILL 1)	010000 RX xxx x01 0
SHIFT R0 RX	SHIFT TO RIGHT (FILL 0)	010000 RX xxx x00 1
SHIFT R1 RX	SHIFT TO RIGHT (FILL 1)	010000 RX xxx x01 1

Input and Output (I/O) instructions

Input

INCHAR RX

RX<-"00000000"&key

110101 | RX | xxx | xxx | x

Output

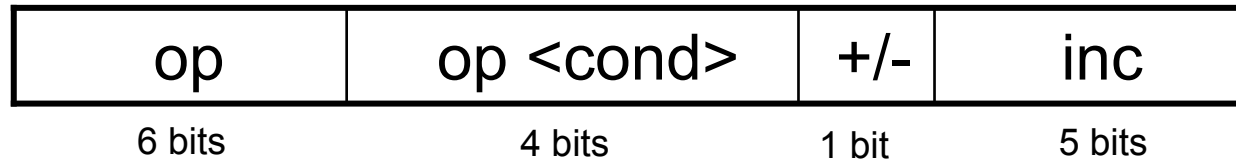
OUTCHAR RX RY

VIDEO(RY)<-CHAR(RX)

110001 | RX | RY | xxx | x

Instruction format

- Jump and Branch (JB-type)



Branch instructions

- Branches only if the condition is true

BRA +/-INC	PC<-PC +/- INC	000001 0000 +/- INC
BEQ +/-INC	PC<-PC +/- INC	000001 0011 +/- INC
BNE +/-INC	PC<-PC +/- INC	000001 0101 +/- INC
BZ +/-INC	PC<-PC +/- INC	000001 1010 +/- INC
BNZ +/-INC	PC<-PC +/- INC	000001 1011 +/- INC
BC +/-INC	PC<-PC +/- INC	000001 0111 +/- INC
BNC +/-INC	PC<-PC +/- INC	000001 1000 +/- INC
BGR +/-INC	PC<-PC +/- INC	000001 0010 +/- INC
BLE +/-INC	PC<-PC +/- INC	000001 0001 +/- INC
BEG +/-INC	PC<-PC +/- INC	000001 0100 +/- INC
BEL +/-INC	PC<-PC +/- INC	000001 0101 +/- INC
BOV +/-INC	PC<-PC +/- INC	000001 1001 +/- INC
BNOV +/-INC	PC<-PC +/- INC	000001 1100 +/- INC

Jump instructions (all with END)

- Jumps to END only if the condition is true

JMP END	PC<-END	000010 0000 x xxxxx END
JEQ END	PC<-END	000010 0000 x xxxxx
JNE END	PC<- END	000010 0101 x xxxxx
JZ END	PC<- END	000010 1010 x xxxxx
JNZ END	PC<- END	000010 1011 x xxxxx
JC END	PC<- END	000010 0111 x xxxxx
JNC END	PC<- END	000010 1000 x xxxxx
JGR END	PC<- END	000010 0010 x xxxxx
JLE END	PC<- END	000010 0001 x xxxxx
JEG END	PC<- END	000010 0100 x xxxxx
JEL END	PC<- END	000010 0101 x xxxxx
JOV END	PC<- END	000010 1001 x xxxxx
JNOV END	PC<- END	000010 1100 x xxxxx

Call instructions (all with END)

- Calls the procedure only if the condition is true

CALL END	MEM(SP)<-PC PC<-END SP--	000011 0000 x xxxxx END
CEQ END	idem	000010 0000 x xxxxx
CNE END	idem	000010 0101 x xxxxx
CZ END	idem	000010 1010 x xxxxx
CNZ END	idem	000010 1011 x xxxxx
CC END	idem	000010 0111 x xxxxx
CNC END	idem	000010 1000 x xxxxx
CGR END	idem	000010 0010 x xxxxx
CLE END	idem	000010 0001 x xxxxx
CEG END	idem	000010 0100 x xxxxx
CEL END	idem	000010 0101 x xxxxx
COV END	idem	000010 1001 x xxxxx
CNOV END	idem	000010 1100 x xxxxx

Return instruction

RTS

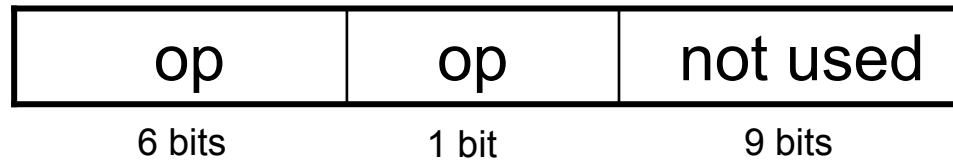
SP++

000100 | xxxx | x | xxxxx

PC<=MEM(SP)

Instruction format

- System control (C-type)



Control instructions

CLEARC	C<-0	001000 0 xxxxxxxxx
SETC	C<-1	001000 1 xxxxxxxxx
HALT	STOP EXECUTION	001111 x xxxxxxxxx
NOP	NO OPERATION	000000 x xxxxxxxxx

Instruction format

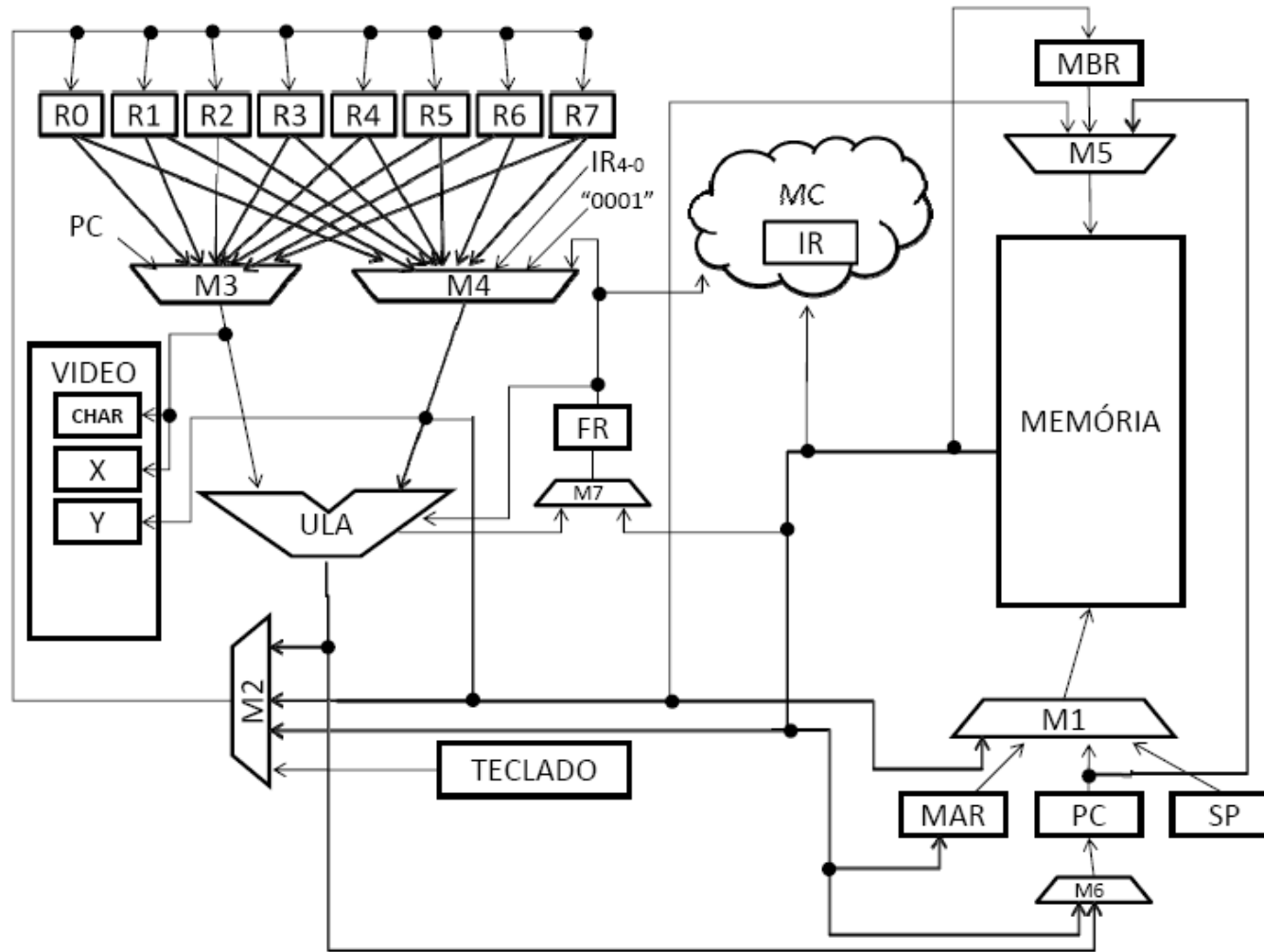
- Stack operations (S-type)



Stack instructions

PUSH RX	MEM(SP) <- RX SP--	000101 RX 0 xxxxxx
PUSH FR	MEM(SP) <- FR SP--	000101 xxx 1 xxxxxx
POP RX	MEM(SP) <- RX SP++	000110 RX 0 xxxxxx
POP FR	MEM(SP) <- FR SP++	000110 xxx 1 xxxxxx

The COMP09 processor microarchitecture



Special registers

- FR:
 - Flag Register
 - 16 bits
 - Bit definition
 - <...|stackunderflow|stackoverflow|DivByZero|ArithmeticOverflow|carry|zero|equal|lesser|greater>

Interpreting Machine Language Code

- Start with opcode
- Opcode tells how to parse the remaining bits

Exercises

- Convert the following SEER assembly code into machine language. Write the instructions in hexadecimal

INCHAR 0

INCHAR 1

OUTCHAR 0 1