Execução de instruções no Y86

O modelo de execução de uma máquina sequencial como o Y86 é extremamente simples. As instruções são executadas umas a uma pela ordem com que aparecem escritas no programa. Alterações a esta ordem têm que ser explícitas, isto é, tem que existir uma instrução que explicitamente altera a ordem de execução; chamamos a estas instruções saltos. No Y86 existem 6 tipos de saltos: saltos incondicionais (`jmp`), saltos condicionais (`jle`, `jl`, `je`, `jne`, `jge`, `jg`), invocação de funções (`call`) e retorno de funções (`ret`).

O processador executa repetidamente um ciclo que pode ser visto como constituído por 5 fases (ver figura na próxima página):

extracção da instrução – A instrução armazenada na posição de memória cujo endereço está no registo Program Counter (PC) é lida da memória para um registo auxiliar designado por Instruction Register (IR); O endereço da instrução imediatamente a seguir à actual é calculado somando o comprimento da instrução ao valor do PC; no Y86 este endereço da próxima instrução é designado por `valP`;

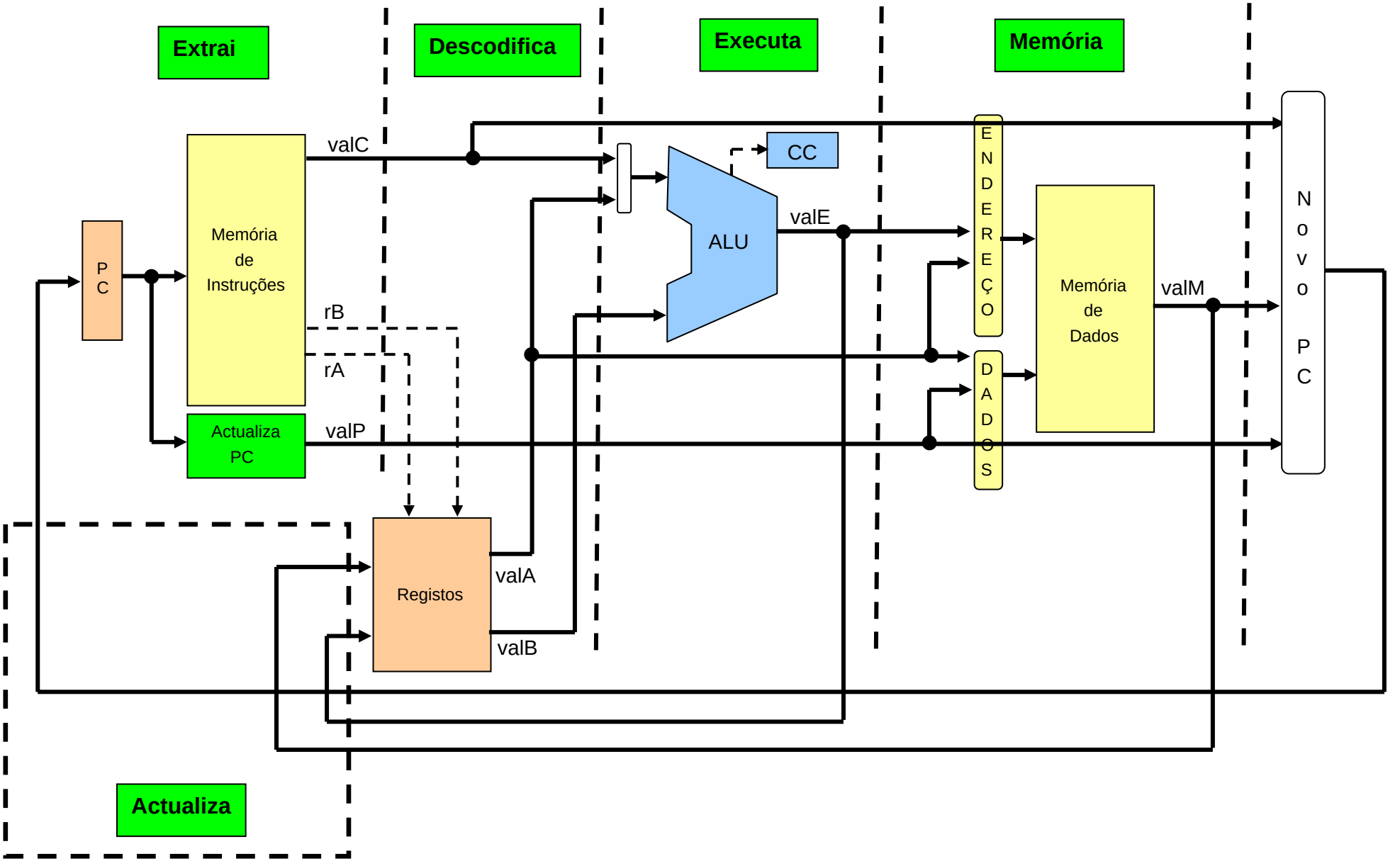
descodificação da instrução e leitura dos operandos – A unidade de controlo descodifica a instrução que se encontra no IR e, simultaneamente, são lidos do banco de registos os operandos desta instrução. No Y86 todos os operandos são valores imediatos ou valores que se encontram armazenados nos registos, podendo, portanto, ser disponibilizados nesta fase; os registos a utilizar são indicados nos campos `rA` e `rB` da instrução; os respectivos valores são colocados pelo banco de registos nos sinais `valA` e `valB` do processador; o sinal correspondente ao valor imediato é referido como `imm`;

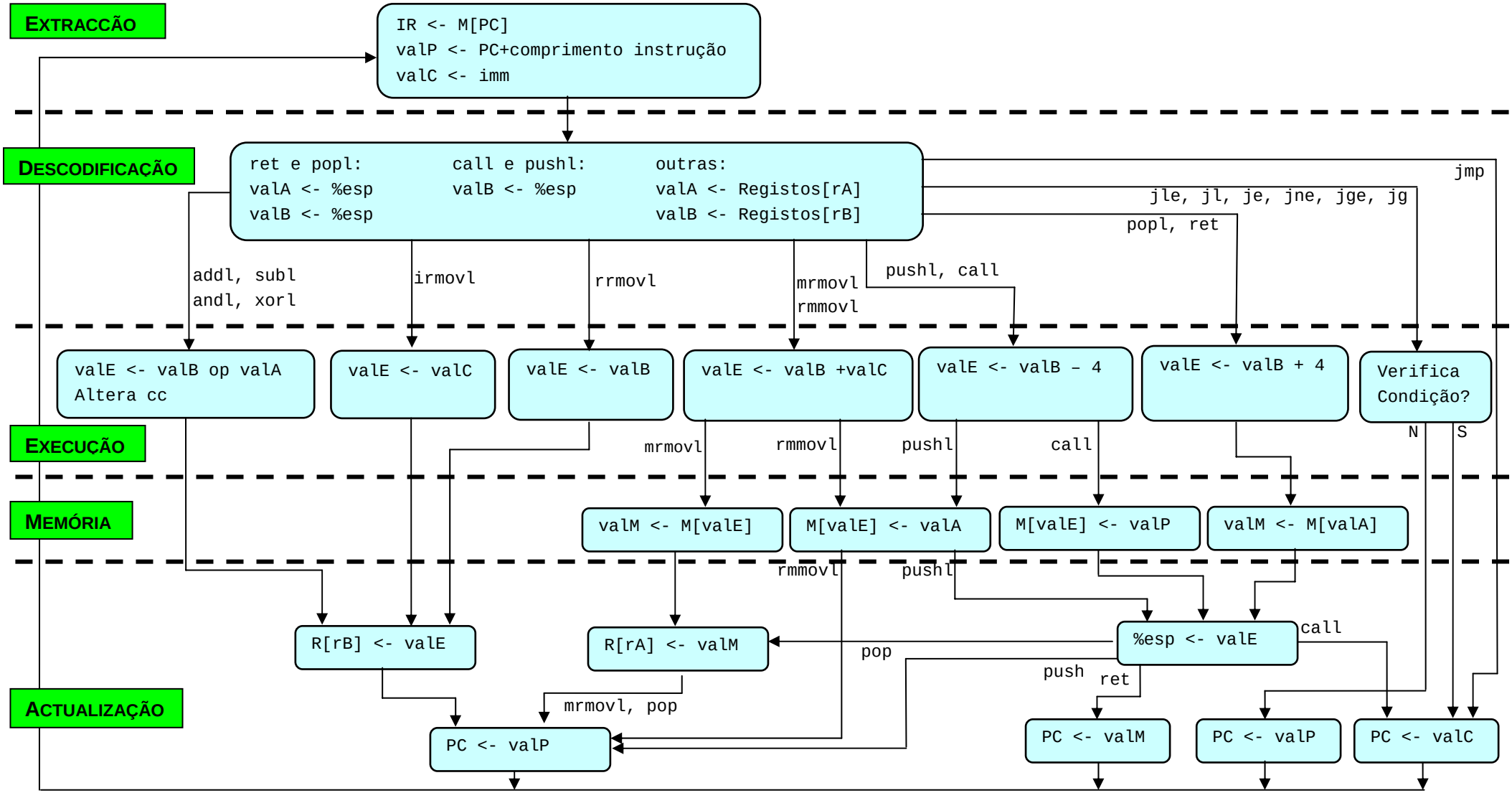
execução – é realizada na Unidade Lógica-Aritmética (vulgarmente designada por ALU – Arithmetic and Logic Unit) a operação correspondente à instrução em causa, alterando os códigos de condição. Esta operação pode ser indicada explicitamente pela instrução (caso de um addl, por exemplo) ou pode ser necessária implicitamente (caso de um acesso à memória onde o conteúdo do registo base deve ser somado ao deslocamento para calcular o endereço a ler ou escrever). O resultado é designado por `valE`. As instruções de salto condicional consultam os códigos de condição (cc, designados por *flags* na terminologia inglesa) para determinar se o salto é tomado;

memória – se a instrução implica um acesso à memória (`mrmovl, rmmovl`) este é realizado nesta fase. De notar que o endereço da posição de memória a ler ou escrever foi calculado na fase anterior; o valor lido é designado por `valM`.

actualização –o resultado da execução da instrução é escrito no registo correspondente. Adicionalmente, o PC é actualizado para apontar para a próxima instrução; esta pode ser a instrução consecutiva (`valP`), ou, no caso de saltos tomados, a instrução indicada como destino do salto.

De realçar que esta subdivisão e ordenamento das fases de execução de instruções é essencialmente conceptual, ajudando a perceber como é que uma instrução deve ser executada. Na realidade, o número e ordem das fases podem ser diferentes.


Extrai
Descodifica
Executa
Memória
Actualiza
PC
Memória de Instruções
valC
rB
rA
valP
ALU
CC
valE
ENDEREÇO
DADOS
Memória de Dados
valM
Novo PC
Registos
valA
valB


EXTRACÇÃO
IR <- M[PC]
valP <- PC+comprimento instrução
valC <- imm
DESCODIFICAÇÃO
ret e popl:
valA <- %esp
valB <- %esp
call e pushl:
valB <- %esp
outras:
valA <- Registos[rA]
valB <- Registos[rB]
jmp
jle, jl, je, jne, jge, jg
popl, ret
addl, subl
andl, xorl
irmovl
rrmovl
mrmovl
rmmovl
pushl, call
EXECUÇÃO
valE <- valB op valA
Altera cc
valE <- valC
valE <- valB
valE <- valB +valC
valE <- valB – 4
valE <- valB + 4
Verifica
Condição?
N
S
MEMÓRIA
mrmovl
rmmovl
pushl
call
valM <- M[valE]
M[valE] <- valA
M[valE] <- valP
valM <- M[valA]
ACTUALIZAÇÃO
R[rB] <- valE
R[rA] <- valM
%esp <- valE
call
pop
push
ret
mrmovl, pop
PC <- valP
PC <- valM
PC <- valP
PC <- valC