Fundamentos de Arquitetura de Computadores

Tiago Alves

Faculdade UnB Gama
Universidade de Brasília

Simulador RARS: chamadas de sistema

Implementa o montador, pseudo-instruções, simula um sistema operacional com funções de Entrada/Saída em console próprio.

Disponível em: `https://github.com/TheThirdOne/rars`

Ex.: Aplicação que escreve na tela: `the answer = 5`

```
1  .data # Tell the assembler we are defining data not code
2  str:  # Label this position in memory so it can be referred to in our code
3    .string "the answer = " # Copy the string
4
5  .text # Tell the assembler that we are writing code (text) now
6  main: # Make a label to say where our program should start from
7
8    li a0, 1   # li means to Load Immediate and we want to load the value 1 into register a0
9    la a1, str # la is similar to li, but works for loading addresses
10   li a2, 13  # like the first line, but with 13. This is the final argument to the system call
11   li a7, 64  # a7 is what determines which system call we are calling and we what to call write (64)
12   ecall      # actually issue the call
13
14   li a0, 5   # li means to Load Immediate and we want to load the value 5 into register a0
15   li a7, 1  # a7 is what determines which system call we are calling and we what to call write (1)
16   ecall      # actually issue the call
17
18
19   li a0, 0   # The exit code we will be returning is 0
20   li a7, 93  # Again we need to indicate what system call we are making and this time we are calling exit(93)
21   ecall
```

Simulador RARS: chamadas de sistema

Name	Call Number (a7)	Description	Inputs	Outputs
PrintInt	1	Prints an integer	a0 = integer to print	N/A
PrintFloat	2	Prints a floating point number	fa0 = float to print	N/A
PrintDouble	3	Prints a double precision floating point number	fa0 = double to print	N/A
PrintString	4	Prints a null-terminated string to the console	a0 = the address of the string	N/A
ReadInt	5	Reads an int from input console	N/A	a0 = the int
ReadFloat	6	Reads a float from input console	N/A	fa0 = the float
ReadDouble	7	Reads a double from input console	N/A	fa0 = the double

Simulador RARS: chamadas de sistema

ReadString	8	Reads a string from the console	a0 = address of input buffer a1 = maximum number of characters to read	N/A
Sbrk	9	Allocate heap memory	a0 = amount of memory in bytes	a0 = address to the allocated block
Exit	10	Exits the program with code 0	N/A	N/A
PrintChar	11	Prints an ascii character	a0 = character to print (only lowest byte is considered)	N/A
ReadChar	12	Reads a character from input console	N/A	a0 = the character
GetCWD	17	Writes the path of the current working directory into a buffer	a0 = the buffer to write into a1 = the length of the buffer	a0 = -1 if the path is longer than the buffer
Time	30	Get the current time (milliseconds since 1 January 1970)	N/A	a0 = low order 32 bits a1=high order 32 bits

Arquiteturas alternativas

Alternativa de projeto:

- forneça operações mais poderosas;
- o objetivo é reduzir o número de instruções executadas;
- o risco é um tempo de ciclo mais lento e/ou uma CPI mais alta.

Vejamos o IA-32!

IA-32

Linha temporal:

- 1978: O Intel 8086 é anunciado (arquitetura de 16 bits);
- 1980: O co-processador de ponto flutuante Intel 8087 é acrescentado;
- 1982: O 80286 aumenta o espaço de endereçamento para 24 bits; disponibiliza mais instruções;
- 1985: O 80386 estende para 32 bits; novos modos de endereçamento;
- 1989-1995: O 80486, Pentium e Pentium Pro acrescentam algumas instruções (especialmente projetadas para um maior desempenho);
- 1997: 57 novas instruções MMX (SIMD) são acrescentadas; Pentium II;
- 1999: O Pentium III acrescenta outras 70 instruções (SSE — Streaming SIMD Extensions): reação aos concorrentes (ADM 3DNow).
- 2001: Outras 144 instruções (SSE2)

IA-32

Linha temporal:

- 2003: A AMD estende a arquitetura para aumentar o espaço de endereço para 64 bits; estende todos os registradores para 64 bits, além de outras mudanças (AMD64)
- 2004: A Intel se rende e abraça o AMD64 (o chama EM64T) e inclui mais extensões de mídia Essa história ilustra o impacto das "algemas douradas" da compatibilidade "adicionando novos recursos da mesma forma que se coloca roupas em uma sacola", uma arquitetura "difícil de explicar e impossível de amar".
- 2006: SSE4 adiciona 54 instruções! Suporte especial a máquinas virtuais.
- 2007: AMD anuncia 170 instruções do conjunto SSE5
- 2011: Intel anuncia Advanced Vector Extension: registradores SSE evoluem de 128 bits para 256 bits, redefinição de 250 instruções e adição de 128 instruções.

Visão geral do IA-32

Complexidade:

- instruções de 1 a 17 bytes (136 bits) de tamanho
- um operando precisa agir como origem e destino
- um operando pode vir da memória
- modos de endereçamento complexos, por exemplo, "índice base ou escalado com deslocamento de 8 ou 32 bits"

Registradores e endereçamento de dados do IA-32

Registradores no subconjunto de 32 bits que surgiram com o 80386

Restrições de registrador do IA-32

Não são viáveis operações memória-memória.

Imediatos podem ser de 8, 16 ou 32 bits. Qualquer um dos registradores (exceto EIP e EFLAGS) se enquadram nas combinações.

Source/destination operand type	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

Restrições de registrador do IA-32

Mode	Description	Register restrictions	MIPS equivalent
Register indirect	Address is in a register.	Not ESP or EBP	`lw $s0,0($s1)`
Based mode with 8- or 32-bit displacement	Address is contents of base register plus displacement.	Not ESP	`lw $s0,100($s1) # <= 16-bit # displacement`

Base plus scaled index	The address is Base + (2^{Scale} x Index) where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,0($t0)`

Base plus scaled index with 8- or 32-bit displacement	The address is Base + (2^{Scale} x Index) + displacement where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,100($t0) #<=16-bit # displacement`

Instruções típicas do IA-32

Quatro tipos principais de instruções de inteiro:

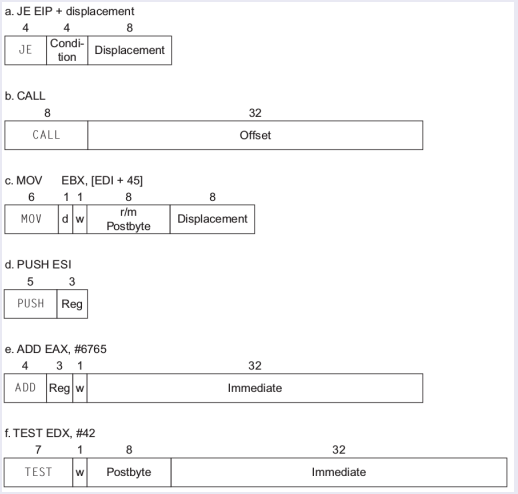
- Movimento de dados, incluindo move, push, pop
- Aritmética e lógica (registrador de destino ou memória)
- Fluxo de controle (uso de códigos de condição/flags)
- Instruções de string, incluindo movimento e comparação de strings.

Instruction	Function
je name	if equal(condition code) {EIP=name}; EIP-128 <= name < EIP+128
jmp name	EIP=name
call name	SP=SP-4; M[SP]=EIP+5; EIP=name;
movw EBX,[EDI+45]	EBX=M[EDI+45]
push ESI	SP=SP-4; M[SP]=ESI
pop EDI	EDI=M[SP]; SP=SP+4
add EAX,#6765	EAX= EAX+6765
test EDX,#42	Set condition code (flags) with EDX and 42
movsl	M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4

call salva EIP da próxima instrução na pilha/stack. EIP é o PC da Intel.

Formatos de instruções do IA-32

Formatos típicos:



Restrições de registrador do IA-32

Instruction	Meaning
Control	**Conditional and unconditional branches**
`jnz, jz`	Jump if condition to EIP + 8-bit offset; `JNE` (for`JNZ`), `JE` (for `JZ`) are alternative names
`jmp`	Unconditional jump—8-bit or 16-bit offset
`call`	Subroutine call—16-bit offset; return address pushed onto stack
`ret`	Pops return address from stack and jumps to it
`loop`	Loop branch—decrement ECX; jump to EIP + 8-bit displacement if ECX ≠0
Data transfer	**Move data between registers or between register and memory**
`move`	Move between two registers or between register and memory
`push, pop`	Push source operand on stack; pop operand from stack top to a register
`les`	Load ES and one of the GPRs from memory

Restrições de registrador do IA-32

Arithmetic, logical	**Arithmetic and logical operations using the data registers and memory**
`add, sub`	Add source to destination; subtract source from destination; register-memory format
`cmp`	Compare source and destination; register-memory format
`shl, shr, rcr`	Shift left; shift logical right; rotate right with carry condition code as fill
`cbw`	Convert byte in eight rightmost bits of EAX to 16-bit word in right of EAX
`test`	Logical AND of source and destination sets condition codes
`inc, dec`	Increment destination, decrement destination
`or, xor`	Logical OR; exclusive OR; register-memory format
String	**Move between string operands; length given by a repeat prefix**
`movs`	Copies from string source to destination by incrementing ESI and EDI; may be repeated
`lods`	Loads a byte, word, or doubleword of a string into the EAX register

Resumo

A complexidade da instrução é apenas uma variável

- instrução mais simples versus CPI mais alta / velocidade de clock mais baixa

Princípios de projeto:

- simplicidade favorece a regularidade
- menor é melhor
- bom projeto exige comprometimento
- agilizar o caso comum

Arquitetura do conjunto de instruções: uma abstração muito importante!