

Componentes de um computador

- memória principal

- armazena as instruções e dados do programa
- palavras de memória → organiza a memória em 8 bits
 - ↳ cada palavra de memória está associada a um endereço
 - ↳ unidade básica de leitura e escrita na memória
- no P15C-V → cada instrução é codificada em 4 bytes e ocupa 1 palavra de memória.

- CPU (unidade central de processamento)

- executa os programas, as instruções, de computador
- busca as instruções do programa na memória principal → lida com, no mínimo, 1 palavra de memória.
- ao executar uma instrução, a CPU também pode ler / escrever na memória
- reguladores → dispositivos de armazenamento de dados ou instruções
 - ↳ PC → guarda o endereço de memória da próxima instrução
 - ↳ IR → instruction register → guarda a instrução lida da memória
- control unit → unidade de controle que organiza o funcionamento
 - ↳ envia comando de leitura para a memória principal com o endereço armazenado em PC
- datapath → realiza operações aritmética e lógicas com os dados e endereços

- memória secundária (persistente)

- armazena os dados e os programas de maneira persistente
- mais lenta → informação é passada para a memória principal para ser usada

- barramento

- sistema de comunicação entre as componentes do computador

- periféricos

- entrada e saída

codificação de programas de computador

- código fonte → linguagem de alto nível

- arquivos texto
- devem ser compilados para serem executáveis

- script → linguagem de alto nível

- arquivos texto
- são executados por outros programas de computador

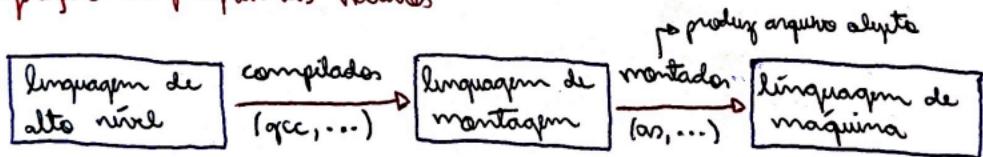
- código binário para arquiteturas virtuais

- linguagem de máquina
- arquivos binários
- sequência de instruções codificadas de forma binária
- executados por outros programas de computador

- código nativo

- linguagem de máquina → pode ser executado diretamente na CPU
- arquivos binários → instruções nativas codificadas de forma binária
- instruções nativas → instruções que a CPU do computador entende

geração de programas nativos



execução de programas nativos

→ como?

- carrega o programa na memória principal
- opera no PC e endereço de memória da 1ª instrução do programa
- BIOS → programa armazenado em uma memória fixa e carrega o SO
 ↳ procura por um boot loader em uma memória secundária (disco) e carrega para a memória principal

representação de informação no computador

- informações são representadas através de dígitos binários (BITS)
- voltagem posicional \rightarrow valor de dígito depende da sua posição
- base numérica
 - o valor de um dígito em um número depende da sua posição
- conversão de bases numéricas

tipos de conversão

decimal \rightarrow binário

decimal \rightarrow hexadecimal

binário \rightarrow decimal

hexadecimal \rightarrow decimal

hexadecimal \rightarrow binário

binário \rightarrow hexadecimal

procedimento usado

divisões por 2 até quo = 0. Cima \leftarrow baixo

divisões por 16 até quo = 0. Cima \leftarrow baixo

soma de potências de 2

soma de potências de 16

expande cada dígito hex em 4 binários

ponta 4 binários e converte em hexa

números com sinal

- todos os bits são utilizados como dígitos de números

sinal e magnitude

- 1º dígito = sinal $\begin{cases} + \rightarrow 0 \\ - \rightarrow 1 \end{cases}$

- próximos dígitos representam o número

complemento de 1

- 1º dígito = sinal

- se for negativo (1º dígito = 1), ~~trocá-lo~~ trocar todos os dígitos para salvo sinal

complemento de 2

- 1º dígito = sinal

- se for negativo (1º dígito = 1), trocar todos os dígitos a excepção do 1º 1.

representação de informação no computador

maior nº	sem sinal	sinal e mag	compl 1	compl 2
menor nº	$2^n - 1$	$2^{n-1} - 1$	$2^{n-1} - 1$	$2^{n-1} - 1$
	0	$-(2^{n-1} - 1)$	$-(2^{n-1} - 1)$	$-(2^{n-1})$

→ IAS

- utiliza 40 bits para codificar cada número
- palavras de memória possuem 40 bits
- registradores da VLA possuem 40 bits

→ arquitetura de 32 bits

- capaz de armazenar e realizar operações aritméticas em números com até 32 bits

→ aritmética binária

- soma → ocorre normalmente, quando passar a limite, vai para a próxima casa
- subtração → quando tivemos algo de tipo 10^{-1} , pegamos empréstimo do dígito a esquerda, o qual fica = 0 e a subtração fica ($2-1 \Rightarrow 1$)
- overflow → quando o resultado é maior (menor) do que a palavra de computador pode representar

→ representação de caracteres

- cada caractér é associado a um número distinto (ASCII = 7 bits, = 128 caract.)
- texto = cadeia de caracteres

→ organização de dados na memória

- Número₁₀ = AAAAAAAA BBBB BBBB CCC CCCC DDDDDDD₂

endereço big-endian little-endian

00	AAAA...	DDDD...
01	BBBB...	CCCC...
02	CCC...	BBBB...
03	DDD...	AAAA...

↓ final = maior endereço
comigo = maior endereço

representação de informação no computador

- vetores na memória

- os elementos do vetor são armazenados de forma consecutiva na memória

- registos na memória (estruturas)

- os elementos da struct são armazenados de forma consecutiva na memória

- matrizes na memória

- row-major order → as linhas da matriz são armazenadas consecutivamente
- column-major order → as colunas da matriz são armazenadas consecutivamente

geração de programas nativos

- compiladores

- converte programas de linguagem de alto nível para linguagem de montagem
- arquivos texto

- montadores

- converte de linguagem de montagem para linguagem de máquina
- arquivos objeto → codificados de forma binária

- ligadores

- liga o código de vários arquivos objeto e produz um arquivo executável

rotulos e símbolos

- rotulos

- marcadores que representam posições no programa / na memória

• montadores e ligadores convertem rotulos em endereços e ajudam a referência

- usados para determinar a posição inicial de variáveis afetivas e de rotinas do programa

- símbolos

- "nomes" que são associados a valores numéricos

• tabela de símbolos → mapeia os nomes dos símbolos aos endereços que representam as posições dos rotulos no programa (montador transforma rotulos em símbolos)

Referências e relocações

- referências

- instruções ou diretivas de um programa em linguagem de montagem podem referenciar símbolos pelo nome
- o montador e o ligador substituem as referências pelo valor de símbolo

- Tabela de relocação

- contém informações sobre ítems do programa (instruções e dados)
- referenciam símbolos e necessitam de ajustes quando as posições dos símbolos mudam
- montador → para cada referência, o montador adiciona uma entrada na tabela
- ligador → usa a tabela de relocação para ajustar as referências
- referências não definidas
 - referências e símbolos que não foram definidos
 - o ligador tenta resolver essa situação olhando para símbolos de outros arquivos

símbolos globais e locais

- símbolos locais → apontam visíveis dentro de um mesmo arquivo
- símbolos globais → visíveis externamente
 - diretiva `.globl nome` transforma "nome" em global

ponto de entrada do programa

- definição → endereço da primeira instrução que deve ser executada quando o programa é iniciado
- endereço é gravado pelo ligador em um campo no cabeçalho do arquivo executável
Lo procurar por `_start` → se não encontrar, vai usar o 1º endereço do arquivo

organização do programa em seções

- ".text" → armazenamento do código do programa (instruções)
- ".data" → variáveis globais inicializadas
- ".bss" → variáveis globais não inicializadas
- ".rodata" → armazenamento de constantes (read-only)
- arquivo objeto
 - montador produz arquivo objeto com seções
 - cabeçalho → indica o tamanho e a posição das partes restantes do arquivo
 - conjunto de seções → .text, .data
 - tabela de símbolos → mantém lista de símbolos definidos ou não
 - tabela de relocação → lista de referências para símbolos
- linkador
 1. agrupa as seções
 2. ajusta/reloca as referências a símbolos
 3. liga os símbolos exportados e não definidos
- diretiva .section
 - ".section sec" → instrui o montador a mudar para a seção sec.
 - ".section .data
X: .word 10
 - ".section .bss
X: .skip 1000
- .function .text
 - função: → define função
 - set endereço, o nº
 - offset

sintaxe da linguagem de montagem

- comentários

- descartados pelo montador → remove comentários e espaços em branco entre linhas
- /* comentário */
- # comentário

- rótulos

- anotações de lugar ou endereço (marcadores)
- montador → converte rótulo em endereço
- rótulo simbólico → convertido para símbolo e adicionado na tabela de símbolos
 - ↳ anota a posição (endereço) de variáveis globais e rotinas do código
- rótulo numérico → rótulos locais úteis para referenciar trechos de códigos próximos
 - ↳ pode ter mais de um no mesmo arquivo
 - ↳ referência: nº b (backward, nº antes) ou nº f (forward, nº depois)

- instruções de montagem

- montador → traduz as instruções de montagem para instruções de máquina
- pseudo-instruções → instruções em linguagem de montagem que não existem em linguagem de máquina
- valores imediatos (imm) → valores numéricos
 - ↳ quando usamos "...", estamos registrando o nº que isso representa em ASCII
- índices → "names" associados a valores numéricos
 - ↳ podem ser usados como parâmetros em algumas diretivas e instruções

- contadores de localização

- contador interno do montador que auxilia no processo de atribuir endereços às instruções
- contém o endereço da próxima posição livre da memória

- diretivas de montagem

- comandos para controlar o processo de montagem

diretivas de montagem

- inserção de valores

- .string } adicionam uma string em ASCII e terminam em 0 (+1 byte)
- .asciz }
- .ascii → adiciona uma string em ASCII, mas nem sempre adiciona o 0
- .byte → adiciona 1 ou + bytes no ponto atual de montagem (.byte 10, byte com valor 10)
- .half → adiciona valor de 16 bits (2 bytes)
- .word → adiciona valor de 32 bits (4 bytes)

- diretiva .section

- informa ao montador qual seção deve receber os próximos elementos a serem montados

- diretiva .align

- no PIV32, as instruções precisam ser armazenadas em endereços múltiplos de 4
↳ os dados podem ser armazenados em endereços ~~múltiplos de 2, 4, 8~~
- montador não verifica se a instrução está ~~ainda~~ alinhada ou não
- .align 2 → alinha as instruções

arquitetura do P15E-V

- arquitetura von Neumann

- memória principal → armazena dados e instruções
- unidade lógica e aritmética → registradores de propósito geral

- arquitetura load / store

- os valores têm que ser carregados nos registradores antes de realizar-se operações

- deslocamento de bits

- slli $a_0, a_2, 2 \rightarrow a_0 = a^2 \cdot 2^2 \rightarrow$ desloca 2 casas para esquerda

• slli $a_1, a_3, 1 \rightarrow a_1 = a_3 / 2 \rightarrow$ desloca 1 casa para direita

- formato little-endian

- byte menos significativo na menor endereço

instruções: movimentações de dados

- word

- int

- unsigned int

- byte

- char

- byte unsigned

- unsigned char

- half word

- short

- half word unsigned

- unsigned short

instruções: controle de fluxo

- instruções capazes de mudar o fluxo normal (ordenado) de execução

↳ ocorre o desvio apenas sob certas condições

- jpl → pl ra, lpc

- invoca rotinas

- opera o endereço da instrução subsequente ($PC + 4$), que é onde deve retornar depois do pl

- ecall → invoca o sistema operacional

- ret

- retorna de função e segue as instruções

- j → apenas pula!

- não salva nenhum endereço de retorno

chamada e retorno de funções

- jal → salta o endereço de retorno
- ret → salta para o endereço de retorno

pilha do programa

- quando uma rotina chama uma subrotina, perdemos o endereço de retorno
- rotina ativa
 - rotina que ainda não retornou → precisa armazenar dados na pilha
 - cresce para baixo → pilha começo do maior endereço e vai diminuindo
- stack pointer → SP → aponta para o topo da pilha
 - addi sp, sp, -4 } empilha o valor
sub a₀, 0(sp)
 - lru a₀, 0(sp) } desempilha o valor
addi sp, sp, 4
 - último empilhado (0(sp)) é o primeiro desempilhado
↳ loop, na é o 1º empilhado ((n-1)*4, sp)
- → tudo que empilha antes de função, desempilha depois da função

ABI: a política de uso dos registradores

- retorno de valores das funções

- até 32 bits (4 bytes) → a₀
- entre 32 bits e 64 bits → a₀ e a₁
 - ↳ - significativo
 - ↳ + significativo

- parâmetros → a₀, a₁, ..., a₇, sendo que o resto vai na pilha

- passagem de parâmetros por valor → li a₀, valor

- passagem de parâmetros por referência → la a₀, endereço de valor

política do uso dos registradores

- callc - salvad

- registradores devem ser salvos pela rotina chamadora (mix)

• $T_0 - T_6$ e $a_0 - a_7$ e ra

- calle - salvad

- registradores que devem ser salvos pela rotina sendo chamada (exchange)

• $D_0 - D_{11}$

- quadro de pilha

- frame pointer $\rightarrow fp \rightarrow$ aponta para o início do quadro atual

- addi $sp, sp, -8$ } aponta para o início do frame
addi $fp, sp, 8$ }

acessando periféricos

- periféricos \rightarrow dispositivos de entrada e saída

- I/O devices (in/out)

- conectados à CPU através de barramentos

- podem possuir registradores ou mámoias internas

↳ operações de entrada e saída são realizadas através da leitura e escrita nestes registradores e memórias internas

- conversão de periféricos com a CPU

- barramentos \rightarrow caminhos de comunicação entre 2 ou + dispositivos

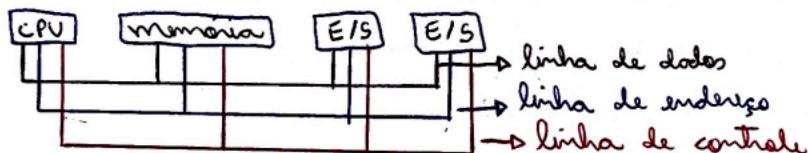
↳ PCI \rightarrow desenvolvido pela Intel

↳ AMBA \rightarrow desenvolvido pela ARM

barramentos

- comunicações

- realizada através do envio de endereços, dados e comandos
- o barramento contém linhas de comunicação (fios) que transmitem a informações
- barramentos com linhas exclusivas
 - cada tipo de informação possui uma linha de comunicação exclusiva



- múltiplos barramentos

- todos os dispositivos podem estar ligados em um mesmo barramento
- ↳ problema → todos têm que operar na mesma velocidade
- barramento local
- barramento do sistema
- barramento de alta velocidade
- barramento de expansão

- leitura e escrita de dados em periféricos

- operações de entrada e saída são realizadas através da leitura e escrita nos registradores e memórias internas dos periféricos

- CPU ↔ periférico

- instruções de ISA
 - { Port-mapped I/O
memory-mapped I/O

- memória principal ↔ periférico (DMA)

Leitura e escrita de dados em periféricos

- port - mapped I/O

- método que emprega instruções especializadas para copiar dados entre a CPU e os periféricos
- I/O port → valor numérico que identifica os registradores e posições de memórias internas dos periféricos
- I/O instruction → instrução especializada para copiar dados entre a CPU e periféricos
- I/O address space → conjunto de valores válidos de I/O ports
- obstruções → espaço de endereçamento de palavras da memória principal e o I/O address space são distintos
↳ CPU diferencia os dois com base na instrução sendo executada

- memory - mapped I/O

- método que faz uso das mesmas instruções para copiar dados entre a CPU e os periféricos e a CPU e a memória principal
- há apenas 1 espaço de endereçamento
- faíscas de endereço são mapeadas na memória principal e em periféricos
- usar lw e sw

- memória principal ↔ periférico (DMA)

- DMA blocker → co-procesador especializado

Técnica de espera ocupada

- busy waiting → espera ocupada → técnica de sincronização na qual o programa aguarda uma condição em loop (verifica repetidamente)
- laço que checa a condição a cada iteração falsa → tenta de novo
- exemplo → busy loop, read - keyboard
 ↳ loop de verificação → segue a execução
 ↳ até ser = 1

interrupções

- matrizes

- alguns eventos que ocorrem em periféricos exigem a atenção imediata da CPU
- assim que uma tecla for pressionada, fazer a CPU copiar o valor do Keypad Controller para um buffer na memória principal
 - ↳ os programas do sistema param a ler teclas do buffer!
- a atenção da CPU deve ser direcionada para tratar o evento do periférico
- abordagens para tratar o evento do periférico
 - { polling
 - interrupções externas

interrupções: polling

- definição → método no qual o programa é projetado para checar os periféricos ~~periodicamente~~ periodicamente

interrupções externas

- definição → sinais enviados por dispositivos externos à CPU para informar que eles precisam de atenção
- CPU para o que está fazendo, "atende" o periférico, e retorna a sua atividade
- antes de tratar a interrupção, é importante salvar todo o "contexto" do programa que está executando
- fluxo de tratamento
 - interrupção acontece → fluxo de controle desviado para a ISR
 - contexto recuperado → interrupção é tratada → contexto é salvo

- fluxo de tratamento

interrupt service routine

- interrupção acontece → fluxo de controle desviado para a ISR
- contexto recuperado → interrupção é tratada → contexto é salvo

- detectando interrupções externas

- A CPU tem 1 ou + pinos de entrada para receber sinais de interrupção
- unidade de controle da CPU monitora os pinos

interrupções externas

- chamando a ISR adequada

- cada periférico que gera interrupção tem sua própria ISR

- abordagem SW-only

- CPU invoca ISR genérica que
 - determina qual periférico interrompeu
 - invoca a rotina de tratamento adequada
- não tem suporte do hardware \rightarrow ISR genérica interage com os periféricos para saber qual interrompeu

- vantagem \rightarrow HW da CPU + simples

- desvantagem \rightarrow desempenho do mecanismo de interrupções ruim (precisa checar todos)

- abordagem SW/HW

- CPU invoca ISR genérica que
 - determina qual periférico interrompeu
 - invoca rotina de tratamento adequada

- HW provê um suporte para descobrir qual foi o periférico causador
 \hookrightarrow exemplo \rightarrow bocado número que identifica o periférico

- vantagem \rightarrow desempenho não depende do N° de periféricos nem da velocidade

- desvantagem \rightarrow HW da CPU + + complexo

- abordagem HW-only

- CPU determina a causa da interrupção e invoca a ISR adequada automaticamente.

- interrupt vector table \rightarrow tabela na memória principal que ajuda a invocar a ISR

- vantagem \rightarrow melhor desempenho entre as abordagens

- desvantagem \rightarrow HW + complexo

interrupções externas no RISC-V

- control and status registers (CSRs)

- registradores especiais que permitem que o software configure/controle a CPU
 ↳ expõem estados internos da CPU para o software

- instruções especiais
 - cmr → copia o conteúdo de CSRs para registradores
 - csrrw → copia o conteúdo de um registrador para o CSR
 - csrrw → troca o conteúdo do registrador com o do CSR

registers

- MIE → habilita ou desabilita o tratamento de interrupções
- MPIE → salva o valor anterior de MIE quando uma interrupção é tratada
- MPTR → modo de operação

cause

- INTEERRUPT → indica se a interrupção foi causada por uma interrupção ou exceção
- EXCCODE → indica a causa da interrupção

mtvec

- BASE → endereço usado para invocar ISAs
- MODE → como identificar o endereço de ISA

- mie.MIE → habilita ou desabilita o tratamento de interrupções

- mip.MEIP → indica se tem interrupção pendente

- mepc → salva o valor do PC antes de setá-lo para o endereço de ISA

- mscratch