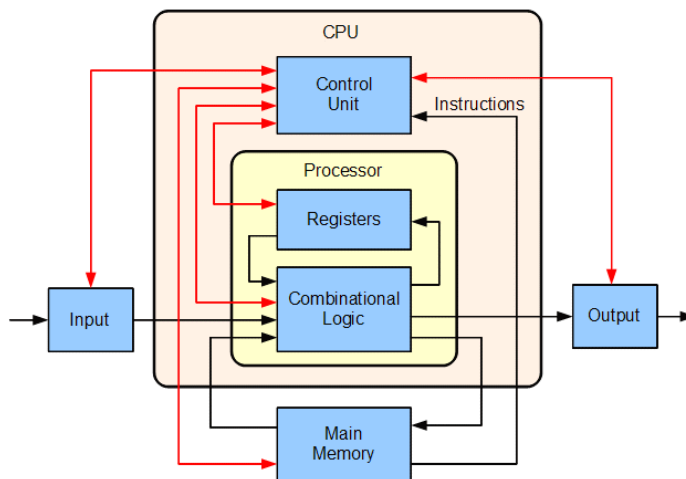


Instituto de Computação
Departamento de Ciência da Computação
Programação de Computadores II
Professor: Anselmo Montenegro
Entrega: 19 de outubro de 2016

Primeiro Trabalho

Um computador é composto de vários componentes como **CPU**, **memória principal**, **dispositivos de entrada** e **dispositivos de saída**. Um processo computacional ocorre através da interação entre os vários componentes do sistema, os dados e o usuário.



Fonte: https://en.wikipedia.org/wiki/Central_processing_unit#/media/File:ABasicComputer.gif

Escreva um programa em C que simule um computador simplificado.

A memória do computador simplificado é caracterizada por um conjunto de palavras de 32bits. A CPU é composta de uma unidade lógica e aritmética capaz de executar um pequeno conjunto de instruções. Além disso, ela possui três registradores: o **contador de programa (pc)** que armazena o endereço da próxima instrução a ser executada; o **contador de instruções (ir)** que armazena a instrução corrente e o **acumulador (acc)** que é um registrador utilizado para dar suporte às operações. As instruções disponíveis encontram-se abaixo, com os respectivos códigos em hexadecimal e suas descrições:

Instrução	Código (hexadecimal)	Explicação	Exemplo de opcode
NVALID	0x0	Instrução inválida	00 00 00 00
READ	0x1	Le uma palavra do dispositivo de entrada (o console) para o acumulador	00 01 00 00
WRITE	0x2	Escreve um valor armazenado no acumulador no dispositivo de saída (o console)	00 02 00 00
LOAD	0x3	Carrega para o acumulador uma palavra da memória no endereço XX especificado pelos 2 bytes menos significativos	00 03 XX XX
STORE	0x4	Armazena na memória, no endereço XX, a palavra contida no acumulador	00 04 XX XX
ADD	0x5	Adiciona o valor no endereço de memória XX ao valor	00 05 XX XX

		no acumulador	
SUB	0x6	Subtrai o valor no endereço de memória XX do valor no acumulador	00 06 XX XX
MUL	0x7	Multiplica o valor no acumulador pelo valor no endereço de memória XX	00 07 XX XX
DIV	0x8	Divide o valor no acumulador pelo valor no endereço de memória XX	00 08 XX XX
MOD	0x9	Determina o resto da divisão do valor no acumulador pelo valor no endereço de memória XX	00 09 XX XX
BRANCH	0xA	Transfere o fluxo de execução para o endereço de memória XX	00 0A XX XX
BRANCHNEG	0xB	Transfere o fluxo para o endereço XX caso o valor no acumulador seja negativo	00 0B XX XX
BRANCHZERO	0xC	Transfere o fluxo para o endereço XX caso o valor no acumulador seja igual à zero	00 0C XX XX
HALT	0xD	Aborta o programa	00 0D XX XX

A versão simplificada do computador executa um programa carregado em memória através do seguinte conjunto de passos:

1. A próxima instrução a ser executada é buscada no endereço identificado pelo *contador de programa* e armazenada no *contador de instruções*.
2. O *opcode* de 32 bits no *contador de instruções* é decodificado, sendo extraída a instrução (*operation*) e o operando (*operand*), caso exista.
3. A instrução é executada.
4. O *contador de instruções* é atualizado conforme o fluxo determinado pela instrução executada.

O sistema deve ser subdividido em três módulos: principal (main.c), cpu (cpu.c, cpu.h), memória (mem.c, mem.h). O módulo principal (main.c) utiliza os demais módulos.

No arquivo cabeçalho(mem.h) do módulo *mem* encontramos a definição do tipo **word_t** que descreve uma palavra da memória e do tipo **memory_t**, que descreve um array de **word_t**. Além disso temos as assinaturas das funções que manipula a memória:

```
#ifndef MEM_H_INCLUDED
#define MEM_H_INCLUDED
#ifdef HUGE
/*Tamanho maximo da memoria.Espaco de enderecamento de 16 bits*/
#define MAX_MEM_SIZE 65536
#else
#define MAX_MEM_SIZE 256
#endif

typedef int word_t;

typedef word_t memory_t[MAX_MEM_SIZE];

/* Limpa todo o conteudo da memoria usada pela CPU */
void clearMem(memory_t mem);

/* Imprime no console todos os enderecos de memoria e seus conteudos */
void dumpMem(const memory_t mem);

/* Retorna uma palavra da memoria no endereco especificado*/
word_t getWord(const memory_t mem, int address);

/* Armazena uma palavra na memoria*/
void setWord(memory_t mem,int address, word_t word);

#endif // MEM_H_INCLUDED
```

O módulo `cpu` deve descrever a CPU como um tipo definido pelo usuário baseado em um tipo estruturado (*struct*) da linguagem C. Observe que a CPU conhece (tem uma dependência) para a memória e retém seu endereço em um dos seus campos.

```
#ifndef CPU_H_INCLUDED
#define CPU_H_INCLUDED

#include "mem.h"

#define NUM_INST 14

enum instruction{INVALID = 0x0, READ = 0x1, WRITE = 0x2, LOAD = 0x3, STORE = 0x4, ADD = 0x5, SUB = 0x6,
    MUL = 0x7, DIV = 0x8, MOD = 0x9, BRANCH = 0xa, BRANCHNEG = 0xb, BRANCHZERO = 0xc, HALT =
    0xd};

typedef enum instruction instruction_t;

struct decode {char * inst_name;
    word_t opcode;} decode;

struct _cpu
{
    memory_t mem;
    word_t acc;
    int pc;
};

typedef struct _cpu cpu_t;

/* Array contendo pares <nome_instrucao, codigo_instrucao> */
extern struct decode decode_list[NUM_INST];

/* retorna um opcode, isto e, uma palavra de memoria com a instrucao e o operando */
word_t getOpcodeCPU(const cpu_t cpu);

/* Inicia o funcionamento da CPU, zera o contador de programa e limpa a memoria */
void init(cpu_t *cpu);

/* Roda um programa previamente carregado na memoria da CPU */
void run(cpu_t *cpu);

void exec(cpu_t *cpu);

/* Sai da cpu */
void exitCPU(const cpu_t cpu);

/* Mostra o conteudo da CPU */
void dumpCPU(const cpu_t cpu);

/* Le um arquivo com um programa em linguagem de montagem e compila gerando um codigo objeto */
void assembleProgram(char *src_file_name, char *obj_file_name);

/* Carrega em memoria os bytes de um programa executável */
void loadProgram(cpu_t *cpu, char *file_name);

#endif // CPU_H_INCLUDED
```

Escreva um programa em C que seja capaz de ler um código na linguagem de montagem “Tinyasm” com extensão .ty e monte os bytecodes gerando o código objeto, que deve ser carregado em memória, antes da execução. Cada *opcode* traduzido é carregado em memória em ordem crescente de endereço.

Exemplo de programa na linguagem de máquina do computador simplificado:

```
LOAD 7 ; carrega o valor no endereço 7 no acumulador
WRITE ; escreve o valor do acumulador
LOAD 8 ; carrega o valor no endereço 8 no acumulador
WRITE ; escreve o valor do acumulador
ADD 7 ; adiciona o valor no endereço 7 no acumulador
WRITE ; escreve o valor do acumulador
HALT ; interrompe a execução
4
5
```

```
LOAD 11 ; acc = n
BRANCHZERO 10 ; se (n=0) salte para
LOAD 12 ; acc = sum
ADD 13 ; acc = acc + inc
STORE 12 ; sum = sum + inc
WRITE ; escreve sum
LOAD 11 ; acc = n
SUB 13 ; acc = acc - inc
STORE 11 ; n = n - 1
BRANCH 0 ; return a primeira instrução
HALT ; termina o programa
3 ; (n = 3)
0 ; (sum = 0)
1 ; (inc = 1)
```