**Disciplina**: Performance em Sistemas Ciberfisicos

**Professor:** Guilherme Schnirmann

**Nome Estudante: Enzo Enrico Boteon Chiuratto**

**Atividade Prática / Relatório**

**Computador IAS**

# **Descrição da Atividade:**

O objetivo dessa atividade é entender como funciona o computador IAS. Esse computador é o primórdio dos computadores atuais, ou seja, é um computador Von Neumann, ainda que com estrutura arcaica é uma excelente ferramenta para entender os fundamentos e características do processador.

A memória do computador IAS é dividida em 4096 palavras (4k = 2¹²). Ou seja, temos uma memória com 12 bits de endereçamento. No nosso simulador o endereçamento está sendo feito em hexa, ou seja, 4 bits para cada dígito. Exemplo:

Posição 0 = 0000 0000 0000 = 000

Posição 10 = 0000 0000 1010 = 00A

Posição 1000 = 0011 1110 1011 = 3EB

Vamos utilizar um simulador desenvolvido na UNICAMP:

Estamos no nível mais baixo da arquitetura, ou seja, aqui as instruções são codificadas em linguagem de máquina. O formato da instrução da arquitetura do computador IAS (em hexadecimal):

**000** **01** **001** **05** **002**

**12 bits 20 bits 20 bits**

**Endereço instrução instrução**

**na memória**

Repare que temos os 3 primeiros dígitos representando o endereço em que as 2 próximas instruções serão armazenadas ao mapear em memória. **Cada dígito é um hexa e representa 4 bits.**

**000** **01** **001** **05** **002**

= **0000 0000 0000** **0000 0001** **0000 0000 0001** **0000 0101** **0000 0000 0010**

**0 0 0** **0 1** **0 0**  **1** **0 5** **0 0 2**

**Mapa memória opcode endereço opcode endereço**

**Atenção: os 3 primeiros dígitos não fazem parte da instrução! Lembre-se que a instrução tem 40 bits (começa no primeiro opcode).**

A seguir algumas instruções (opcodes) básicas:

**LOAD (01): carrega valor do endereço de memória no AC: AC <-M(X)**

**STOR (21); escreve valor do AC no endereço de memória M(X) <- AC**

**ADD (05); soma valor do endereço de memória no AC: AC <- AC +M(X)**

**SUB (06); subtrai valor do endereço de memória no AC: AC <- AC -M(X)**

**MUL (0B); multiplica valor do endereço de memória no MQ: <- MQ\*M(X)**

**LOAD MQ (mem.) (09); Carrega valor da memória para MQ: MQ <-M(X)**

**LOAD MQ AC (0A); Carrega valor de MQ para AC: AC<-MQ**

**DIV (0C). Divide valor de AC por valor de endereço da memória e resultado vai para MQ e resto para AC: MQ<-AC/M(X)**

**JUMP M(X, INSTRUÇÃO ESQUERDA) – (0D) – O Program Counter salta para a instrução à esquerda da palavra na memória armazenada no endereço M(X).**

**JUMP M(X, INSTRUÇÃO DIREITA) – (0E) – O Program Counter salta para a instrução à direita da palavra na memória armazenada no endereço M(X).**

**JUMP+ M(X, INSTRUÇÃO ESQUERDA) – (0F) – Se AC>=0 então PC <-M(X). Salta para a instrução à esquerda da palavra de memória se o valor armazenado em AC for maior ou igual a zero.**

**JUMP+ M(X, INSTRUÇÃO DIREITA) – (10) – Se AC>=0 então PC <-M(X). Salta para a instrução à direita da palavra de memória se o valor armazenado em AC for maior ou igual a zero.**

**M(X) é o endereço que será o “parâmetro” na instrução do opcode.**

Para utilizar o simulador, deve-se atribuir na memória as instruções em hexadecimal. **Exemplo**:

**005 00 000 00 002** (valor atribuído em memória no endereço 005) **-DADO**

**006 3** (valor atribuído em memória no endereço 006) **-DADO**

**000 01 005 05 006;**

**- 000**: endereço de atribuição em memória (mapeamento)

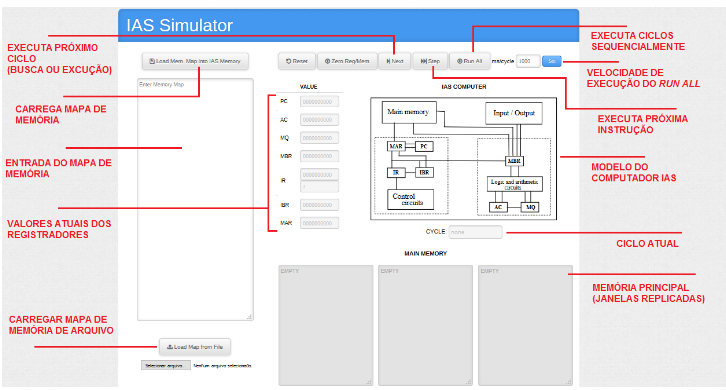
- **01**: instrução **LOAD** em hexadecimal;

- **005**: Endereço de memória de que vai ser feito o LOAD

-**05**: instrução ADD em hexadecimal

- **006**: endereço de memória de que vai ser feito o ADD

**Simulador:**



**Entrega:**

Esta atividade deverá ser entregue até o final da aula no Canvas.

O estudante deverá entregar um arquivo “.pdf” contendo as respostas da atividade proposta no roteiro.

**Roteiro da Atividade: Nessa atividade vamos conhecer as estruturas de salto dentro da memória principal. Fique atento aos comandos de JUMP adicionados na nossa lista no começo do arquivo:**

**JUMP M(X, INSTRUÇÃO ESQUERDA) – (0D) – O Program Counter salta para a instrução à esquerda da palavra na memória armazenada no endereço M(X).**

**JUMP M(X, INSTRUÇÃO DIREITA) – (0E) – O Program Counter salta para a instrução à direita da palavra na memória armazenada no endereço M(X).**

**JUMP+ M(X, INSTRUÇÃO ESQUERDA) – (0F) – Se AC>=0 então PC <-M(X). Salta para a instrução à esquerda da palavra de memória se o valor armazenado em AC for maior ou igual a zero.**

**JUMP+ M(X, INSTRUÇÃO DIREITA) – (10) – Se AC>=0 então PC <-M(X). Salta para a instrução à direita da palavra de memória se o valor armazenado em AC for maior ou igual a zero.**

Atente-se para copiar o código em “Enter Memory Map” e clicar em “load Mem. Map into IAS Memory”. **Sempre que fizer uma alteração no seu código, você vai precisar resetar os registradores e carregar novamente o mapa de memória.**

**Atenção: só coloque os prints que forem necessários para explicar o entendimento da lógica, se conseguiu explicar com suas palavras o que está sendo feito, um print com resultado de funcionamento está suficiente.**

1. Acesse o simulador IAS: <https://www.ic.unicamp.br/~edson/disciplinas/mc404/2017-2s/abef/IAS-sim/>
2. Revisando a prática anterior: Implemente no computador IAS (você escolhe as posições de memória de dados que irá utilizar).

a = 16

b = 4

c = 20

d = 5

x = a/b + c/d

Interface gráfica do usuário, Aplicativo, Tabela

Descrição gerada automaticamente

O programa armazena o valor de 00A no acumulador, chama a função 0C para dividir o valor de 00B pelo valor do acumulador (16d / 4d).

O valor da equação, armazenada no registrador MQ é passado para o acumulador utilizando a função 0A (load), logo então usamos a função 21 (Store) para armazenar o valor na variável 00E (4d)

Realizamos o mesmo processo com a equação seguinte (20d/5d), porém ao invés de armazenar o valor em uma variável, guardamos o resultado da equação no acumulador

*003 0A 000*

e adicionamos o valor do acumulador ao valor armazenado em 00E, nossa variável com o valor da equação anterior (4d)

*05 00E*

Assim, nosso acumulador agora tem o valor da equação, 8d

1. Implemente no computador IAS:

a = 5

b = 3

c = 20

d = 4

x = (a\*b + c/a)/d

000 09 00A 0B 00B //load 5->mq | mul mq -> 3

001 0A 000 21 00E //mq -> ac | ac -> 00E

002 01 00C 0C 00D // load 20 -> ac | div ac -> 4

003 0A 000 05 00E // mq -> ac |ac-> ac + 00E

004 0C 00D 00 000 // div ac -> 4 || 00 000

00A 00 000 00 005

00B 00 000 00 003

00C 00 000 00 014

00D 00 000 00 004

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

1. Considerando o exemplo:

000 01 10B 05 10C ***//000 LOAD M(10B) ; ADD M(10C)***

001 21 10B 0D 000 ***//001 STOR M(10B) ; JUMP M(000, ESQ.)***

10B 00 000 00 002 ***//DADO 002 CARREGADO EM MEMÓRIA NA POSIÇÃO 10B***

10C 00 000 00 001 ***//DADO 001 CARREGADO EM MEMÓRIA NA POSIÇÃO 10C***

Execute passo-a-passo (clicando em **next**) e explique com suas palavras o que está sendo executado e o fluxo das informações no computador IAS.

Interface gráfica do usuário

Descrição gerada automaticamente com confiança média

O valor 02d (10C) é carregado ao AC, logo então, 1d (10C) é adicionado ao valor de AC

Gráfico, Gráfico de caixa estreita

Descrição gerada automaticamente

Logo então, o valor de 10B é atualizado com o valor da soma no registrador AC, e o programa volta para o início com o jump 0D 001, assim formando um loop infinito que adiciona em 1 o valor em 10B

1. Considere o seguinte exemplo:

000 01 11D 06 11E

001 21 11D 0F 000

11D 00 000 00 005

11E 00 000 00 001

* 1. Indique em cada linha o que significa as informações em memória. É instrução? É dado? O que significa cada grupo de dígitos em cada linha?

As 2 primeiras linhas são instruções, onde na primeira linha o valor de 11D é armazenado no AC, logo então o valor do acumulador é subtraído pelo valor em 11E. Na segunda linha armazenamos o valor em AC para 11D, e realizamos um jump para a esquerda da linha 000, caso o AC seja maior ou igual a 0

Nas 2 linhas seguintes armazenamos 05d em 11D e 01d em 11E

* 1. Antes de simular, traduza esse código de máquina, o que está sendo feito e qual o resultado esperado? Explique com suas palavras o que é executado nesse programa

O programa faz um loop de 5 à 0, totalizando 6 iterações, onde cada uma causa a subtração da variável de controle em 1, o resultado esperado é de que o programa rode 6 vezes até o AC se tornar 0

* 1. Como ficaria esse código em uma linguagem de alto nível? (qualquer linguagem).

Class Main{

Public static void main(String[] args){

Int I = 5;

While(I >= 0){

I --

}

}

}

1. Considere o seguinte exemplo. Os endereços 00A, 00B e 00C são nossas variáveis X, Y e Z, respectivamente.

000 01 00A 06 00B

001 0F 003 01 00B

002 21 00C 0D 004

003 01 00A 21 00C

00A 00 000 00 002 // X = 2

00B 00 000 00 003 // Y = 3

00C 00 000 00 000 // Z = 0

* 1. Antes de simular, traduza esse código de máquina, o que está sendo feito e qual o resultado esperado?

O código armazena o valor de 00A (2d) no acumulador, subtrai o valor em AC pelo valor no endereço 00B (3d), assim resultando em -1d, logo então, se o valor em AC for >= 0, o programa pula para a linha 3, caso contrário, carrega o valor de 00B (3d) no acumulador

Então o valor do acumulador (atualmente 03) é armazenado na variável 00C, e o programa pula para a linha 004, qual não existe, assim finalizando o programa, esperamos que o programa finalize aqui com o valor em 00C sendo 003

Na linha seguinte, carregamos o valor de 00A no acumulador novamente e o armazenamos em 00C

* 1. Explique com suas palavras o que é executado nesse programa

O programa realiza uma soma com uma condicional, onde se a condicional for atendida, o programa carrega o valor do acumulador pós subtração à 00C, caso contrário, o programa é finalizado

* 1. O que acontece se mudarmos os valores para X = 5 e Y = 3?

A condicional *001 0F 003* é atendida, o acumulador se torna 5d (valor de 00A) e armazena o valor à 00C

* 1. Como ficaria esse código em uma linguagem de alto nível? (qualquer linguagem).

function main():void{

let ac = 0;

let x = 2;

let y = 3;

let z:number;

ac -= y;

if(ac >= 0){

x = 2;

}else{

Ac = y

Z = ac;

os.exit()

}

}