

Universidade Federal de Pernambuco - UFPE

### Centro de Informática - CIN

Infraestrutura de Hardware – if674cc

Relatório de Projeto

Recife – 2018.1

**Infraestrutura de hardware – if674cc**

**Especificação de Projeto**

Documento desenvolvido pelo grupo de hardware sob orientação do monitor  *Lucas Miranda Lin (lml),* sob orientação do professor Dr. Adriano Sargento.

Escrito por:

*Daniel Henrique Amorim Correia (dhac),*

*Jefferson Carlos Lima da Costa (jclc),*

*Juliano Cezar Teles Vaz (jctv),*

*Luiz Eduardo Pessoa de Freitas (lepf)*.

# Sumário

|  |  |
| --- | --- |
| *1.Introdução……………………………..……………………………..…………………...* | *4* |
| *2.Unidade de processamento………………………..……………………..…..………....* | *5* |
| *3.Descrição das entidades………………………..…………….……..………..…………* | *6* |
| *4.Descrição dos estados de controle…………………………………………..………...* | *13* |
| *5.Conclusões………………………..……………………………..…..…....………………* | *16* |

# 1.Introdução

O Curso de Infraestrutura de Hardware visa dar uma visão geral dos componentes de um computador, quais sejam: processador, sistema de memória (memória principal e memória cache), dispositivos de entrada e saída e, barramentos. Neste caso, este projeto tem como propósito a implementação de um processador de arquitetura multiciclo básica, ou seja, uma unidade de processamento (CPU) com todos os seus funcionamentos. Além disso, com o objetivo de ter uma ideia de como a CPU funcionaria caso chegasse a ser realmente desenvolvida, simulamos tudo que foi projetado com uso da linguagem de descrição de hardware, verilog, fazendo uso da do Quartus v 9.1. De uma forma geral, o objetivo é fazer com que o estudante passe a entender, para além das aulas e do material científico base recomendado, os diversos aspectos acerca do projeto, organização e implementação de computadores aos que tange as arquiteturas de processadores. E com o relatório tentamos descrever como tudo da CPU foi desenvolvido, para que assim haja um entendimento do projeto sem grandes problemas.

# 

# 2. Unidade de processamento

Foi entregue a cartolina.

# 3. Descrição das entidades

**Entidade div:**

**Entrada**:

Clk(1bit): representa o Clock do sistema

Reset(1bit): representa uma entrada, caso seja setada para 1 o reset é iniciado.

iniciar (1bit): controle da operação

sinal (1bit): Sinal do resultado

Dvdendo (32bits): Dividendo

Divsor (32bits): Divisor

**Saída:**

logic[31:0] quociente, resto, output logic dividindo.

quociente(32bits) : quociente , que vai para mflo

resto(32bits) : resto , que vai para mfhi

dividindo(1bit) : usado para saber se ainda está dividindo ou não.

**Objetivo:** Realizar a operação divisão.

**Algoritmo:** Semelhante à multiplicação, a divisão utiliza 3 registradores internos como base: Q, A e M. O dividendo é carregado nos registradores A e Q como um número em complemento a dois de 2n(64) bits. O divisor por sua vez, também complemento a dois, é carregado no registrador M. Após o carregamento dos registradores internos, caso o DivIn indique o início da operação de divisão, o valor de M sendo 0, é disparada a flag de divisão por zero(DivZero) e o algoritmo não é continuado, para os demais valores de M, o procedimento seguinte repete-se por n(32) vezes: Os registradores A e Q deslocam-se juntos para a esquerda. Caso O sinal de A e M seja igual, faz-se A = A - M, caso contrário, faz-se A = A + M. Se o sinal de A se mantiver após a operação com o M (considerando que o zero não indica mudança de sinal) ou se A e Q forem iguais a zero, o bit menos significativo de Q é setado 1. Se nenhuma dessas condições se aplicar, o bit menos significativo de Q é setado 0 e o valor de A é restaurado (o valor anterior à operação de adição/subtração com o M).

**Entidade mult:**

**Entrada:**

Clk(1bit) : represanta o Clock do sistema

Reset(1bit) : representa uma entrada, caso seja setada para 1 o reset é iniciado.

iniciar(1bit) : controle da operação

sinal(1bit): Serve para saber se o sinal do resultado vai ser positivo ou negativo.

**Saída:**

saida(64bits) : saída do resultado

multiplicando (1bit) : usado para saber se ainda deve multiplicar ( pois consome vários ciclos).

**Objetivo:** Realizar operação de multiplicação.

**Algoritmo:** algorítmo da multiplicação binária sinalizada de Booth.

**Entidade Complemento 2:**

**Entrada:**

entrada(32bits) : entrada do complemento2 , o valor que vai ser complementado a 2

**Saída**

saida(32bits) : saída do complemento2, depois de ter complementado.

**Objetivo:** conseguir ter valores positivos e negativos.

**Algoritmo:** para obter o Complemento 2 de um número obtemos primeiro o complemento de um (invertendo os bits) e depois somamos 1 ao resultado.

**Entidade concat\_jmp:**

**Entrada:**

offset(28bits): offset da instrução

pc\_3128(4bits): pc

**Saída:**

Conct\_out(32bits): saída, com a concatenação completa

**Entidade conversor:**

**Entrada:**

primeiro(32bits): instrução de 32bits, que será convertida para littleEndian ou BigEndian (dependendo do tipo da instrução)

**Saída:**

saida(32bits), instrução terá a saída convertida.

**Entidade desloEsq28:**

**Entrada:**

entrada26(26bits): entrada de 26bits que vai ser concatenado e receber um <<2

**Saída:**

saida28(28bits) : é o resultado da concatenação com 28'b0 e após ser feito um << 2 (shiftleft)

**Entidade desloEsq32:**

**Entrada:**

entrada32(32bits): dado de 32 bits de entrada

**Saída:**

saida32(32bits) : dado de 32bits de entrada, após ter recebido um << 2 (shiftleft duas vezes)

**Entidade Sinal Estendido:**

**Entrada:**

Address(16bits): endereço de 16 bits, que vai ser estendido para 32 bits

**Saída:**

AdressExt(32bits) : a saída, que será o Address estendido para 32bits.

**Objetivo:** Faz a extensão de 16 bits para 32 bits levando em conta o sinal.

**Algoritmo**: Caso o bit na posição [15] seja 1, concatena com 16 uns, se o bit for 0, concatena-se com 16 zeros.

**Entidade Controle:**

**Entrada:**

Clk(1bit): Clock do sistema.

Reset(1bit): entrada do reset, caso se queira “reiniciar” o processador

overflow(1bit): indicar caso tenha overflow

menor(1bit): usado para confirmação, se é menor.

maior(1bit): usado para confirmação, se é maior.

igual(1bit): usado para confirmação, se é igual.

multiplicando(1bit): usado para saber se o módulo mult ainda está processando a multiplicação.

dividindo(1bit): usado para saber se o módulo div ainda está processando a divisão.(demora alguns ciclos)

opcode(6bits): opcode da instrução

funct(6bits): funct da instrução

instr(26bits): parte da instrução

**Saídas:**

Estado(6bits): identificação do estado atual (61 estados, logo é necessário 6 bits)

mudDadEscr(4bits): seleciona o que vai escrever no reg

muxALU(3bits): controla o mux perto da ALU

controleALU(3bits): controle para selecionar a Operação da Alu

Shift\_regDes(3bits): controla o tipo de shift que vai ser realizado no registrador de deslocamento.

muxPC(3bits): mux que seleciona o que vai pra memória

muxRegEscr(3bits):Controle escrita registrador

muxN(3bits): mux que seleciona o que vai para memória

muxB(2bits): mux que seleciona o que vai para memória

muxMemData(2bits): Controle do mux

muxZero(2bits): Controle do mux

muxA(2bits): Controle do mux

muxDes(1bit): Controle do mux

muxMfhi(1bit): Controle MFHI

muxMflo(1bit): Controle do MFLO

WR(1bit): 0 para ler e 1 para escrever na memória

loadRegALU(1bit): escreve no registrador, quando setado em 1

loadRegA(1bit): escreve no reg A, quando setado em 1

loadRegB(1bit):, escreve no Reg B, quando setado em 1

loadRegMemData(1bit): escreve do Reg para Mem

loadRegEPC(1bit):escreve em EPC

escreverIR(1bit): escreve no registrador de instrução (Caso setado em 1)

escreverPC(1bit): controle para escrever no pc

escrevePCcond(1bit): condição para escrever pc

regBanco(1bit): Controle banco reg

loadRegMfhi(1bit): escreve MFHI em REG

loadRegMflo(1bit): escreve MFLO em REG

iniciarMult(1bit): inicia o Mult

iniciarDiv(1bit): inicia o Div

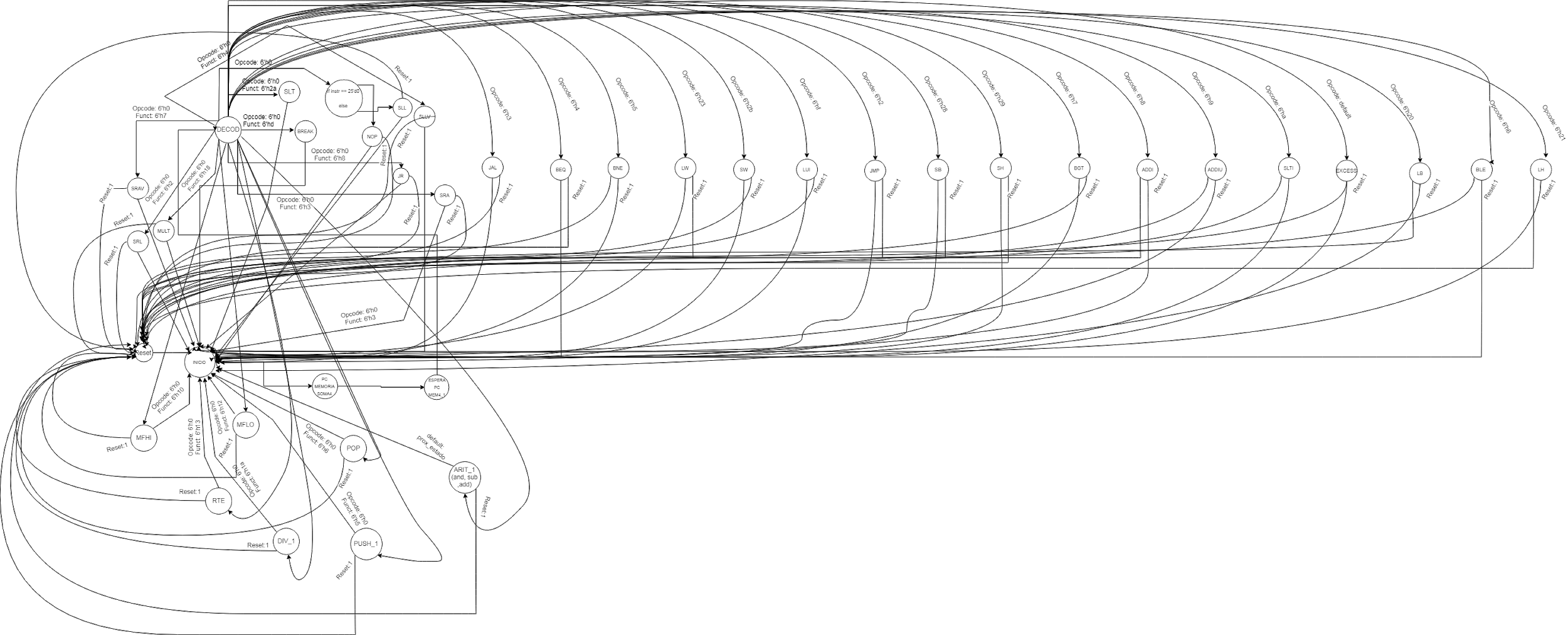
**Objetivo:** unidade criada para controlar todo fluxo do projeto. É ela que irá indicar quais valores serão passados de certas entidades para certas entidades a partir das instruções.

**Algoritmo:** o algoritmo do controle é bem simples, ele é baseado em comparações com jumps. Ele começa em uma label, faz uma comparação (seja de opcode,funct,....) e pula para outra label ou simplesmente o estado não precisa de nenhuma comparação pois só segue um fluxo, pula para o estado do seu fluxo.pula para outra label ou simplesmente o estado (que seria a label) não precisa de nenhuma comparação pois só segue um fluxo, pula para a label do seu fluxo.

# 

# 

# 4. Descrição dos estados de controle

Estado: RESET:

No estado reset limpa-se o conteúdo de todos os registradores do banco de

registradores e aponta o registrador SP para a posição de início da pilha. Os

registradores especiais da CPU são zerados, assim como os registradores internos,

temporários ou não, e saídas de cada uma das demais entidades da CPU.

Estado: PC\_MEMORIA\_SOMA4: Realiza pc+4

Estado: ESPERA\_PC\_MEM4\_1: É um “wait”.

Estado: DECOD: É realizada a decodificação e permite a gravação de pc+4 em pc

Estado: BREAK: Loop em si mesmo.

Estado: SLT\_1:

Estado: SLT\_2:

Estado: INICIO: Ao ser dado o reset, ou ao iniciar uma instrução vai fazer a busca (fetching)

Estado: SHIFT\_1:

Estado: SHIFT\_2:

Estado: JR:Neste estado o conteúdo do registrador A é carregado no PC.

Estado:MUL\_1: O multiplicador carrega os valores dos registradores necessários e inicia a multiplicação dos mesmos.

Estado:MFHI: O conteúdo do registrador MFHI é salvo no registrador indicado no campo rd da instrução atual.

Estado:MFLO: O conteúdo do registrador MFLO é salvo no registrador indicado no campo rd da instrução atual.

Estado:RTE:Vai obter a última instrução feita do EPC e carregar em PC.

Estado:PUSH\_1:Descrementa registrador SP em - 4.

Estado: PUSH\_2:Armazena conteudo de rt no novo SP.

Estado: PUSH\_3:

Estado: POP\_1:Armazena conteudo de SP em rt

Estado: POP\_2:Incrementa registrador SP em +4

Estado: POP\_3:

Estado:ESPERA\_POP:

Estado: POP\_4:

Estado: POP\_5:

Estado: POP\_6:

Estado:DIV\_1:

Estado:ARIT\_1:

Estado:ARIT\_2:

Estado:ARIT\_3:

Estado: JAL: Concatenamos as instruções [25-21] com [20-16] e com [15-0] do registrador de instruções, realizamos um shift left dois estendido de 26 para 28 bits e concatenamos com os 4 bits mais significativos do registrador PC e salvamos essa informação no registrador PC. Paralelamente, o endereço da próxima instrução, atualmente no registrador PC, é salvo no registrador 31 do banco de registradores.

Estado:SW\_1:

Estado: BLE\_1:

Estado: BGT\_1:

Estado:ARIT\_IMM\_1: operação add, controle libera e envia os comandos para o mux e para a ALU.

Estado:ARIT\_IMM\_2:Efetuar a operação.

Estado:ARIT\_IMM\_3:Vai Guardar o resultado em rd.

Estado:EXCES\_1: Faz pc - 4 ( pra pegar a instrução que causou a exceção) via ULA.

Estado: EXCES\_2:guarda valor em EPC.

Estado: EXCES\_3:Selecionar endereco de memoria para cada caso de exceção.

Estado:RTE: Escreve o conteúdo do registrador especial EPC no PC.

Estado: LUI\_1: Vai obter o "imediato",e o imediato vai ser “shiftado” 16 bits à esquerda.

Estado: LUI\_2:Vai guardar o resultado em rt.

Estado:BEQ\_1:Vai obter os dados de rs e rt.

Estado:BEQ\_2:Efetuar a operação na ULA.

Estado:BNE\_1:Vai obter os dados de rs e rt.

Estado:BNE\_2:Efetuar a operação na ULA.

Estado:BLE\_1:Vai obter os dados de rs e rt.

Estado:BLE\_2:Efetuar a operação na ULA.

Estado:BGT\_1:Vai obter os dados de rs e rt.

Estado:BGT\_2:Efetuar a operação na ULA.

Estado:JMP: Faz (pula) com que o PC seja escrito com o valor do endereço alvo

Estado:SW\_1:

Estado:SW\_2:

Estado:ESPERA\_SH\_SB:

Estado: SW\_3:

Estado:SH: Neste estado os dois bytes mais significativos do conteúdo do registrador indicado em rt é concatenado aos 2 bytes menos significativos do conteúdo do registrador MDR e posteriormente, esses 4 bytes resultantes são escritos na memória no endereço [rs + offset] ( sendo rs o conteúdo do registrador do banco de registradores indicado no campo rs da instrução atual).

Estado: SB: Neste estado o byte mais significativo do conteúdo do registrador indicado em rt é concatenado aos 3 bytes menos significativos do conteúdo do registrador especial MDR e posteriormente, esses 4 bytes resultantes são escritos na memória no endereço [rs+offset] (sendo rs o conteúdo do registrador do banco de registradores indicado no campo rs da instrução atual).

Estado: LW\_1:

Estado: LW\_2:

Estado: ESPERA\_LW:

Estado: LW\_3:

Estado:LB:

Estado: LH: No estado LH os dois bytes menos significativos do conteúdo recém-lido na memória são estendidos para 32bits e escritos no registrador especial MDR.

Estado: LW\_4:

Estado: MUL\_2: Esse estado é mantido enquanto o controle aguarda o final da multiplicação, que deve ser sinalizado pela saída do módulo mult. e quando finaliza, o resultado da multiplicação é salvo nos registradores.

Estado: DIV\_1:

Estado: DIV\_2:

# 

# 5. Conclusões

Devido a necessidade de transmitir informação, melhorar desempenhos de instrumentos computacionais, tornar os projetos e suas organizações mais tangíveis às variáveis do mercado e da tecnologia atual que temos, os pesquisadores da área de computação vêm procurando desenvolver métodos para a produção sistemas de software e hardware cada vez mais eficazes e eficientes.  
Nessa mesma perspectiva, compreendendo essa necessidade profissional e acadêmica, e com o conhecimento adquirido ao longo das aulas sobre o funcionamento de uma CPU, Unidade Central de Processamento, desenvolvemos a nossa própria CPU para que pudéssemos compreender de maneira veemente a construção de um processador simples (e com uma arquitetura simples).   
Deste modo, a atividade prática fez com que nós, alunos, passássemos a entender os diversos aspectos de projetos e implementação de computadores de maneira mais real ao que se relaciona os projetos de infraestrutura de hardware. Além de compreendermos que este conhecimento irá nos auxiliar em tarefas da nossa vida profissional, devido a alta demanda de sistemas embarcados e por aspectos como escalabilidade e rentabilidade desse tipo de tecnologia, e irá servir como base para outras disciplinas da área de hardware.