Relatório - Mico XII

Millena Suiani Costa
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
millena.costa@ufpr.br

***Resumo*—O trabalho em questão consistiu no desenvolvimento de um microprocessador, denominado Mico XII, e seus componentes. Foram obtidos como resultado o seu circuito principal – bem como os subcircuitos utilizados em sua constituição –, os hexadecimais das Memórias de Controle e Instruções, além de seu código teste desenvolvido na linguagem *assembly*.**

***Index Terms*—Microprocessador, MIPS, assembly.**

I. Introdução

O presente trabalho foi elaborado e desenvolvido com base no capítulo 10 do livro do professor Roberto A. Hexsel [1], esse que descreve detalhadamente os processos necessários no desenvolvimento de um microprocessador, baseado nos fundamentos de um MIPS32. Através desse, foi possível projetar um circuito principal – e seus devidos subcircuitos – para a implementação do referido processador, esse, por sua vez, denominado Mico XII. Todos os circuitos foram construídos no *software Digital* e levando em conta a sua documentação em português [2]. Além desses, foram desenvolvidos dois códigos hexadecimais que objetivaram a obtenção de um parâmetro de teste para a execução de cada uma das operações das memórias ROM contidas no projeto, além da certificação de um correto funcionamento dessas.

II. Circuito de Dados

A elaboração do trabalho teve início através do planejamento do circuito de dados do Mico XII, esse que define a união de cada um dos componentes que o constituem. Nesse foram inseridos um *Instruction Pointer* (IP), um somador para incremento do IP, multiplexadores, extensores de sinal, distribuídores de *bits*, túneis para uma melhor organização do circuito, um comparador de valores ligado a um *Flip-Flop* tipo D para propagação de sinal positivo no caso da solicitação de execução da instrução *branch if equal* (beq), duas memórias ROM – sendo essas uma Memória de Controle e uma Memória de Instruções – e uma memória RAM – sendo essa a Memória de Dados –, além de uma Unidade Lógica e Aritimética (ULA) e um Bloco de Registradores, sendo ambos os últimos componentes, subcircuitos.

A. Bloco de Registradores

O Bloco de Registradores consiste em 16 registradores de 32 *bits*, sendo esses o registrador r0 – que, independente do valor que se queira gravar neste, apenas armazena o valor zero – e os registradores r1..r15 que poderão receber demais valores.

O funcionamento do bloco se dá através de duas portas A e B, ambas de 32 *bits*, que dispõem dos conteúdos contidos nos registradores endereçados por a e b (portas essas de 4 *bits*). Esses conteúdos podem ser tanto utilizados nas operações de Lógica e Aritimética – cujo resultado destas pode ser armazenado em um registrador endereçado por c através de uma porta de entrada C, sendo esses respectivamente de 4 e 32 *bits* – quanto utilizados nas operações de controle presentes na Memória de Controle.

B. Unidade Lógica e Aritimética

A Unidade Lógica e Aritimética consiste em um subcircuito no qual estão constadas oito operações a serem realizadas com os valores imputados nas entradas A e B, e armazenados nos registradores endereçados por a e b (ambos de 32 *bits*). O resultado a ser armazenado dentre essas operações é escolhido através de um mux com 3 *bits* de seleção (fun) alocados na Memória de Controle e pode ser destinado ao banco de registradores, sendo, assim, armazenado no registrador endereçado por c.

III. Acesso às Memórias

O circuito do Mico XII dispõe de duas memórias ROM, a Memória de Controle – com 16 endereços de 12 *bits* de dados – que armazena os *bits* de controle para cada operação do microprocessador, e a Memória de Instruções – com endereços de 32 *bits* – na qual é armazenado o código de teste do Mico XII.

Para desenvolver o hexadecimal da Memória de Controle, foi estruturada a Tabela I com o nome de cada operação e os valores binários utilizados pelos *bits* de seleção de cada uma das operações e, por fim, traduzido o binário resultante para hexadecimal. Na referida tabela, o binário da operação é apresentado, da esquerda – bit mais significativo – para a direita – bit menos significativo –, além de seus *bits* de *Don't Care* (x) terem sido substituídos por 0 no binário final.

Já para o hexadecimal à ser inserido na Memória de Instruções foi desenvolvido, à princípio, um código em *assembly* contendo todas as operações para teste, os registradores utilizados em cada uma delas e, quando foi necessário, o valor da constante. Se decorreu, assim, a tradução desses para o hexadecimal, convertendo o nome das operações para seus respectivos *opcodes*, os nomes dos registradores para seus endereços a serem alocados em a, b e c, e o valor da constante em decimal para hexadecimal.

Tabela I
DISTRIBUIÇÃO DE BITS NA MEMÓRIA DE CONTROLE I

	proxIP	escR	selB	fun	selC	escM	show	parar
add	00	1	0	000	00	0	x	0
sub	00	1	0	001	00	0	x	0
mult	00	1	0	010	00	0	x	0
and	00	1	0	011	00	0	x	0
or	00	1	0	100	00	0	x	0
xor	00	1	0	101	00	0	x	0
slt	00	1	0	110	00	0	x	0
not	00	1	0	111	00	0	x	0
addi	00	1	1	000	00	0	x	0
ld	00	1	1	000	01	0	x	0
st	00	0	1	000	x	1	x	0
show	x	x	x	000	x	x	1	0
j	10	1	x	000	10	0	x	0
jal	10	1	x	000	10	0	x	0
jr	11	0	x	000	x	0	x	0
beq	01	0	0	000	x	0	x	0
halt	x	x	x	x	x	x	x	1

Tabela II
DEFINIÇÃO DOS BINÁRIOS PARA O CÓDIGO DE TESTE

	Assembly	Operação	a	b	c	cont
L1	addi r1, r0, 4	1000	0000	0000	0001	0000 0100
L2	addi r2, r0, 7	1000	0000	0000	0010	0000 0111
L3	slt r3, r1, r2	0110	0001	0010	0011	0000 0000
L4	beq r0, r3, L7	1110	0000	0011	0000	0011 0000
L5	mul r1, r1, r2	0010	0001	0010	0001	0000 0000
L6	j L8	1100	0000	0000	0000	0011 1000
L7	sub r1, r1, r2	0001	0001	0010	0001	0000 0000
L8	not r1, r1	0111	0001	0000	0001	0000 0000
L9	show r1	1011	0001	0000	0000	0000 0000
L10	and r4, r1, r2	0011	0001	0010	0100	0000 0000
L11	or r6, r2, r4	0100	0010	0100	0110	0000 0000
L12	st 2(r2), r6	1010	0010	0110	0000	0000 0010
L13	ld r5, 2(r2)	1001	0010	0000	0101	0000 0010
L14	xor r5, r5, r6	0101	0110	0101	0110	0000 0000
L15	addi r7, r0, 136	1000	0000	0000	0111	1000 1000
L16	jal r7, L8	1100	0000	0000	0111	0011 1000
L17	add r6, r6, r3	0000	0110	0011	0110	0000 0000
L18	show r6	1011	0110	0000	0000	0000 0000
end	halt	1111	0000	0000	0000	0000 0000

IV. CASOS OMISSOS - BRANCH IF EQUAL (BEQ)

A instrução beq foi elaborada inserindo no circuito um comparador de dois valores que direciona 1 à saída caso ambos sejam iguais e 0 caso contrário. Essa saída foi ligada diretamente à um bit de seleção de um mux que define as seguintes condições: Caso os valores sejam iguais, o seletor receberá o valor de saída 1, fazendo com que o valor de const (endereço da instrução desejada) seja transmitido ao IP, resultando, assim, na operação de salto. Caso contrário, ou seja, com valores diferentes, o seletor receberá o valor de saída 0, e consequentemente, o IP passa a armazenar o endereço da próxima instrução presente em addIP (IP+1), seguindo diretamente para a próxima execução sem realizar o salto.

V. PROGRAMA TESTE

Para o desenvolvimento do programa teste foi planejado um código genérico em *assembly* que realiza todas as operações pré-estabelecidas na memória de controle. Em seguida, para melhor visualização e sintetização das informações, foi estruturada a tabela II, essa que, por sua vez, contém o *assembly* de cada linha do código, seus respectivos opcodes e os registradores e constantes utilizados nessas. Por fim, dispondo de cada um dos binários das instruções de teste, foi possível a tradução desses para o hexadecimal e alocação para teste diretamente na Memória de Instruções.

Na Tabela II a representação dos *bits* é feita, respectivamente da esquerda (bit mais significativo) pra direita (bit menos significativo); Ademais, na coluna *const* são representados apenas os 8 *bits* menos significativos dentre os 16 *bits* de constante pois foram os únicos que sofreram alteração no código de teste. Assim, consideremos os *bits* presentes na tabela como os *bits* de 0..7, e que apresentam o valor 0 os *bits* de 8..15.

VI. CONCLUSÃO

A realização do projeto se deu através das bases fornecidas e seguindo as instruções apresentadas para tal, assim, foi possível o desenvolvimento de um circuito funcional (assim como os subcircuitos que o constituem) e do dump em hexadecimal das memórias nele contidas. Sua funcionalidade foi testada através de um código *assembly* convertido para hexadecimal, focado na verificação das funções projetadas exclusivamente para o Mico XII. Ao fim do projeto concluiu-se que o trabalho finalizado apresentou resultados satisfatórios, executando as instruções solicitadas de maneira correta.

VII. GLOSSÁRIO

As abreviações *bit* ou *bits* advém de *Binary Digit(s)*, em tradução livre do inglês, Digitos Binários.

A abreviação *opcode* advém de *Operational Code*, em tradução livre do inglês, Código Operacional.

Ambas expressões foram amplamente utilizadas na principal referência utilizada como base para o desenvolvimento do presente trabalho, o livro de Roberto A. Hexsel [1].

REFERÊNCIAS

[1] R. A. Hexsel, *Sistemas Digitais e Microprocessadores*. UFPR, 2012, ch. 10.

[2] "Digital - some minor improvements and bug fixes (doc-portugues.pdf)," Helmut Neemann, 02 2022. [Online]. Available: https://github.com/hneemann/Digital/releases/tag/v0.29