Universidade Federal da Bahia (UFBA)
Instituto de Matemática e Estatística (IME)
Departamento de Ciência da Computação (DCC)
MATA 38 - Projetos de Circuitos Lógicos - Prof. George Lima
2018.2 - 23/10/2018 - Especificação de Trabalho Prático

Resumo

Este texto descreve uma arquitetura simplificada e especifica o trabalho prático da disciplina MATA38. Esta arquitetura, denominada ARQ38, foi especialmente projetada para subsidiar a aplicação de conceitos vistos em classe ao mesmo tempo em que serve para introduzir conceitos básicos sobre arquitetura de computadores.

Apesar das suas simplificações, a arquitetura ARQ38 é completamente funcional no sentido em que, com ela, pode-se executar programas completos, mas pequenos, que operam sobre números inteiros de 8 bits. Entre as simplificações realizadas, as principais são: reduzida memória para armazenar dados e instruções; apenas 16 instruções; duas memórias independentes com endereçamento absoluto para facilitar o endereçamento de dados e instruções; nenhuma instrução para manipular pilhas; e as fases busca-decodificação-execução de instruções podem ser realizadas em apenas um ciclo de relógio.

O objetivo do trabalho é construir os circuitos que compõem a máquina ARQ38. O trabalho é dividido em três fases. Na primeira, a Unidade Lógica Aritmética será construída. Na segunda, o Banco de Registradores é integrado à arquitetura. A terceira fase tem como foco o projeto da Unidade de Controle e sua integração com os demais componentes da CPU. Os pesos de cada uma destas fases são 30%, 30% e 40%, respectivamente. A implementação da arquitetura será feita usando o simulador Logisim (`http://www.cburch.com/logisim/`). Após a implementação no Logisim, será possível executar programas reais na máquina construída, o que ilustra o funcionamento interno do computador. Exemplos de trechos de código para a arquitetura ARQ38 serão fornecidos.

1 Descrição geral da arquitetura ARQ38

O computador pode ser visto como sendo um grande conjunto de circuitos digitais interligados, cada um com funções especializadas e trabalhando de forma coordenada e sincronizada. Pode-se dizer que os principais componentes do computador são a unidade central de processamento (CPU), a memória e os periféricos. A CPU é a responsável pela execução das instruções e pela coordenação das operações necessárias para executá-las. A memória é onde residem o conjunto de instruções a serem executadas e os dados usados por tais instruções. Os periféricos interligam o mundo externo ao interior da máquina. Estes componentes são formados por diversas outras unidades, cada uma das quais construídas por circuitos lógicos, combinacionais e sequenciais.

Uma das avaliações da disciplina MATA38 será um trabalho prático, cujo objetivo é construir uma pequena CPU. Este trabalho tem o intuito de (a) desmistificar a complexidade interna da arquitetura de computadores, (b) consolidar conceitos vistos durante o curso, e (c) aumentar o interesse da turma por tópicos relacionados à organização interna de computadores, o que será tratado em detalhes em outras disciplinas do curso. O trabalho é dividido em fases, cada uma com foco em algum componente da CPU. A última fase contempla a interligação de todos os componentes da CPU, produzindo um pequeno computador funcional. Será possível, por exemplo, executar pequenos programas em linguagem Assembly na máquina construída.

A arquitetura da máquina a ser construída chama-se ARQ38 e foi especialmente especificada para este trabalho. A definição de ARQ38 contempla os principais componentes de uma arquitetura de computador real simplificando aspectos mais complexos. Sua estrutura está representada na Figura 1. A arquitetura é composta pela CPU e duas unidades de memória, uma para instruções e outra dedicada aos dados. A CPU é constituída de um Contador de Programa (PC), Registrador de Instrução (IR),

Unidade Lógica e Aritmética (ALU), um Banco de Registradores e a Unidade de Controle (CTLU) com seu Decodificador de Instruções (IDEC). Cada um destes componentes é descrito resumidamente a seguir:

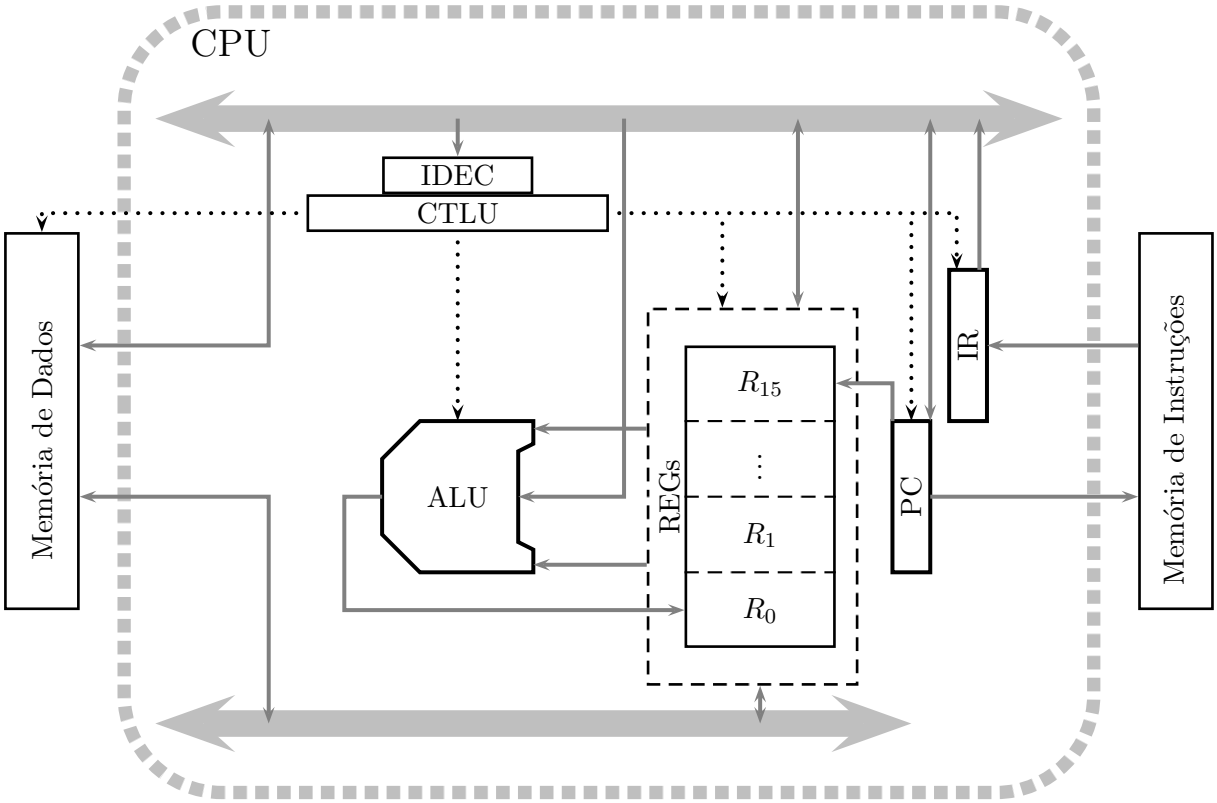

Figura 1: Diagrama funcional da arquitetura ARQ38. As linhas pontilhadas correspondem à ligação com a Unidade de Controle e as sólidas indicam o fluxo de informação dentro da CPU. A arquitetura possui um banco de 16 registradores de propósito geral, duas unidades de memória independentes, para dados e instruções, e registradores especiais. Os registradores PC e IR referem-se, respectivamente, ao Contador de Programa e Registrador de Instruções.

- O Banco de Registradores (REGs) é composto por 16 registradores de propósito geral, cada um com capacidade de armazenamento de 8 bits. Os registradores são identificados como $R_0, \ldots, R_{15}$. O registrador R_0 tem ainda algumas funções especiais. Ele acumula os resultados de todas as operações lógicas e aritméticas e serve, para algumas instruções, como operando implícito. A semântica relacionada a R_0 ficará mais clara quando o conjunto de instruções da arquitetura ARQ38 for descrita. O registrador R_{15} é também usado como apoio à instrução de desvio, como será visto na descrição de tal instrução.

- A Memória de Instrução (IM) armazena as instruções que serão executadas pelo computador e a Memória de Dados (DM) armazena os dados manipulados pelas instruções. As memórias da arquitetura ARQ38 possuem 8 bits de endereçamento e, portanto, contém 2^8 localidades de endereços. Cada localidade da IM tem capacidade de armazenando de 12 bits (1,5 Byte) enquanto que na DM apenas 8 bits (1 Byte) pode ser armazenado em cada localidade.

- A Unidade Lógica Aritmética (ALU) é a unidade responsável pela execução das operações lógicas e aritméticas. Tal componente pode ser visto como vários circuitos combinacionais operando em paralelo, cada um responsável por uma determinada operação.

- A Unidade de Controle (CTLU) gera os sinais de controle responsáveis por garantir o correto funcionamento da CPU e sua conexão com as unidades de memória. Como a IM é apenas de leitura, a arquitetura ARQ38 não faz uso de sinais de controle. Co relação à DM, é necessário indicar qual operação deseja-se realizar na memória, se leitura ou escrita e, portanto, a ligação com a CTLU é necessária. Os sinais de controle são enviados em função do código da instrução. Para tanto, há o decodificador de instruções (IDEC) estritamente ligado a CTLU.

- O PC e o IR são registradores de controle e estado. Na arquitetura ARQ38 estes registradores tem tamanho 8 e 12 bits, respectivamente. O PC é responsável por garantir a busca sequencial das instruções. Após a busca de uma instrução, o PC deve ser incrementado. O IR armazena a instrução a ser executada.

2 Conjunto de Instruções

As instruções da arquitetura ARQ38 podem ser aritméticas e lógicas, de movimentação de dados e de controle. A Tabela 1 fornece a lista e a descrição de cada instrução. Qualquer uma destas instruções possui tamanho de 12 bits. Os quatro bits iniciais de cada instrução (os mais significativos) correspondem ao seu identificador, chamado de código de instrução. Em seguida há 8 bits para operandos, que podem ser identificador de registradores, endereços de memória ou operando imediato. As instruções podem ser classificadas de acordo com seu formato, como descrito a seguir.

2.1 Formato das instruções

- Tipo I. A maioria das instruções da arquitetura ARQ38 são deste tipo. Estas são instruções que possuem dois operandos, cada um dos quais endereçando um registrador. Seu formato é INST R_x, R_y, com x e y representando o identificador dos registradores cujos conteúdos contêm um respectivo operando:

$$\underbrace{b_{11}\, b_{10}\, \cdots b_{08}}_{\text{cod. op.}}\ \underbrace{b_{07}\, b_{06}\, \cdots b_{04}}_{\text{end. reg. } R_x}\ \underbrace{b_{03}\, b_{02}\, \cdots b_{00}}_{\text{end. reg. } R_y}$$

  Um exemplo deste tipo de instrução seria a instrução 'ADD R_1, R_5', cuja representação em hexa é '04015' e em binário é '010000010101'. Note que os primeiros 4 bits identificam a instrução ADD (ver Tabela 1), os quatro bits seguintes indicam que um dos operandos da soma está contido no registrador R_1 enquanto o segundo operando está no registrador R_5.

- Tipo II. Este tipo opera sobre apenas um registrador. Os seus últimos quatro bits (menos significativos) são ignorados. A única instrução deste tipo na arquitetura ARQ38 é a NOT. Seu formato é NOT R_x:

$$\underbrace{b_{11}\, b_{10}\, \cdots b_{08}}_{\text{cod. op.}}\ \underbrace{b_{07}\, b_{06}\, \cdots b_{04}}_{\text{end. reg. } R_x}\ \underbrace{b_{03}\, b_{02}\, \cdots b_{00}}_{\text{bits ignorados}}$$

- Tipo III. Há três instruções deste tipo, LDA, STR e JMP. A primeira lê da memória o conteúdo de um determinado endereço enquanto a segunda armazena num endereço de memória um dado conteúdo. Estas duas instruções possuem dois operandos, um implícito e outro explícito. O operando explícito é o endereço contido nos 8 bits menos significativos que seguem o código da instrução. O operando implícito é o registrador R_0. A instrução JMP possui como operando explícito o endereço de memória para o qual haverá o desvio. Para a instrução JMP, há dois operandos implícitos, R_0 e R_{15}. O conteúdo de R_0 é usado para verificar se o conteúdo do PC deve de fato ser modificado. O registrador R_{15} é usado para guardar o valor atual do PC antes do desvio.

$$\underbrace{b_{11}\, b_{10}\, \cdots b_{08}}_{\text{cod. op.}}\ \underbrace{b_{07}\, b_{06}\, \cdots b_{00}}_{\text{end. de memória}}$$

- Tipo IV. Apenas a instrução LDI é deste tipo. Esta recebe como operando o conteúdo a ser armazenado em R_0.

$$\underbrace{b_{11}\, b_{10}\, \cdots b_{08}}_{\text{cod. op.}}\ \underbrace{b_{07}\, b_{06}\, \cdots b_{00}}_{\text{conteúdo imediato}}$$

- Tipo V. A única instrução deste tipo na arquitetura ARQ38 é 'HLT', que não possui parâmetros e, portanto, tem seus 8 bits menos significativos ignorados pelo processador. Sua função é encerrar o fluxo de controle. Todo programa contém esta instrução como a última a ser executada.

$$\underbrace{b_{11}\, b_{10}\, \cdots b_{08}}_{\text{cod. op.}}\ \underbrace{b_{07}\, b_{06}\, \cdots b_{04}\, b_{03}\, b_{02}\, \cdots b_{00}}_{\text{bits ignorados}}$$

2.2 Especificação do conjunto de instruções

Na arquitetura ARQ38 existem 10 instruções relacionadas à ALU, três para movimentação de dados de/para a unidade de memória DM, uma para movimentação de dados entre registradores, e duas para controle e desvio de fluxo de execução. Das 10 instruções da ALU, três realizam operações lógicas (AND, OR e NOT), quatro são operações aritméticas (ADD, SUB, MUL e DIV) e três efetuam comparações (EQ, LT e LEQ). O resultado destas operações sempre são armazenadas no registrador R_0. Por exemplo, a operação 'ADD R_1, R_2' soma o conteúdo do registrador R_1 com o conteúdo de R_2 e armazena o resultado em R_0. As três instruções para movimentação de dados entre CPU e memória são LDA, LDI, STR. A instrução MOV é usada para transferir dados entre registradores. Por fim, alteração de fluxo de execução podem ser realizados pelas instruções JMP e HLP.

O conjunto completo das 16 instruções e seus respectivos comportamentos é resumido na Tabela 1 e melhor definido a seguir. A seguinte notação é usada nesta descrição: R_x indica o registrador cujo identificador é x; o conteúdo do registrador R_x é representado por $[R_x]$; e o conteúdo de endereço de memória x é dado por $[x]$.

- Instruções lógicas. As operações 'AND R_x, R_y' e 'OR R_x, R_y' realizam operações lógicas bit-a-bit sobre os dois operandos, colocando em R_0 o resultado destas operações. Em outras palavras, para 'AND', tem-se que

$$[R_0] \leftarrow [R_x] \text{ e bit-a-bit } [R_y]$$

  e para 'OR'

$$[R_0] \leftarrow [R_x] \text{ ou bit-a-bit } [R_y]$$

  A operação 'NOT R_x' coloca em R_0 1 caso o conteúdo de R_x seja 0 e coloca 0 caso seu valor seja diferente de 0:

$$[R_0] \leftarrow 1 \text{ se } [R_x] = 0 \quad \text{ e } [R_0] \leftarrow 0 \text{ se } [R_x] \neq 0$$

- Instruções aritméticas. As instruções aritméticas, 'ADD R_x, R_y', 'SUB R_x, R_y', 'MUL R_x, R_y' e 'DIV R_x, R_y', tem comportamento semelhante e operam apenas sobre números inteiros. Elas colocam em R_0 o resultado da operação correspondente:

$$\begin{aligned}
[R_0] &\leftarrow [R_x] + [R_y] \\
[R_0] &\leftarrow [R_x] - [R_y] \\
[R_0] &\leftarrow [R_x] \times [R_y] \\
[R_0] &\leftarrow [R_x] \div [R_y]
\end{aligned}$$

  Na arquitetura ARQ38 não há indicação quando o resultado das operações aritméticas extrapolam a capacidade do registrador. Em outras palavras, não há indicação de *overflow*. Além disso, o resultado da operação de divisão refere-se apenas ao quociente da divisão inteira.

- Instruções de leitura e escrita. A leitura de dados para a CPU pode ser realizada pelas operações 'LDA end' e 'LDI valor'. A primeira coloca em R_0 o conteúdo do endereço de memória correspondente ao seu operando 'end':

$$[R_0] \leftarrow [\text{end}]$$

enquanto a segunda coloca em R_0 o valor do operando 'valor':

$$[R_0] \leftarrow \text{valor}$$

Para armazenar algum conteúdo na memória, usa-se a instrução 'STR end'. Esta coloca o conteúdo de R_0 no endereço dado como parâmetro:

$$[end] \leftarrow [R_0]$$

O conteúdo de um registrador é copiado para outro pela instrução MOV. Para 'MOV R_x, R_y', tem-se a seguinte especificação:

$$[R_x] \leftarrow [R_y]$$

- Instruções de alteração de fluxo de execução. A instrução JMP é responsável por modificar o conteúdo do registrador PC de forma a alterar o fluxo de execução de um programa. Este desvio é condicional e ocorre apenas se o conteúdo de R_0 é igual a 1. Antes do desvio, o valor atual do PC é guardado no registrador R_{15}. A especificação para 'JMP end' é dada por

$$\begin{aligned} \text{Se } [R_0] = 1, [R_{15}] &\leftarrow [PC] \\ [PC] &\leftarrow \text{end} \end{aligned}$$

A instrução HLT, por sua vez, pára o fluxo de execução.

2.3 Exemplo de programa na Arquitetura ARQ38

As seções anteriores apresentaram o conjunto de instruções da ARQ38, seus formatos e especificações. Esta seção oferece uma pequena ilustração de como tais instruções compõem um programa a ser executado. Para tanto, considere o seguinte código em C, exibido na Figura 2. Este é traduzido pelo compilador para um código de máquina correspondente. O compilador aloca espaço de memória para as variáveis e programa. Por ser mais fácil acompanhar esta tradução em linguagem Assembly, o código em C da Figura 2, quando compilado para a arquitetura ARQ38, deve gerar algo como o ilustrado na Figura 3. Aqui considerou-se neste exemplo apenas o código referente à instrução `for`, desprezando-se todos os detalhes da linguagem C e suas bibliotecas. A intenção é apenas oferecer uma pequena ilustração para relacionar a semântica da linguagem Assembly da arquitetura ARQ38 com aquela associada a uma linguagem de alto nível como C.

Como pode ser observado, uma instrução em linguagem de alto nível pode ser traduzida para várias instruções em Assembly. Seria interessante passar algum tempo analisando o trecho de código em Assembly e perceber algumas peculiaridades relacionadas à arquitetura ARQ38. Por exemplo, a implementação de laços condicionais é realizada por operações lógicas e de desvio e ficam bem menos claras quando comparada a programas escritos em linguagens de alto nível. Outras características podem ser ressaltadas, como alocação de variáveis em memória e/ou registradores, movimentações de valores entre registradores, semântica empregada na arquitetura ARQ38 para implementação de desviio condicional etc. É interessante escrever pequenos trechos de código em Assembly da arquitetura ARQ38 para obter certa familiaridade com a linguagem.

Tabela 1: Conjunto de instruções da arquitetura ARQ38.

Operação	Código	Mnemônico	Especificação
E bit-a-bit	00	AND R_x, R_y	$[R_0] \leftarrow [R_x] \,\&\, [R_y]$
OU bit-a-bit	01	OR R_x, R_y	$[R_0] \leftarrow [R_x] \,\|\, [R_y]$
Não lógico	02	NOT R_x	$[R_0] \leftarrow 1$ (se $R_x = 0$); $[R_0] \leftarrow 0$ (se $R_x \neq 1$)
Adição	03	ADD R_x, R_y	$[R_0] \leftarrow [R_x] + [R_y]$
Subtração	04	SUB R_x, R_y	$[R_0] \leftarrow [R_x] - [R_y]$
Multiplicação	05	MUL R_x, R_y	$[R_0] \leftarrow [R_x] \times [R_y]$
Divisão	06	DIV R_x, R_y	$[R_0] \leftarrow [R_x] \div [R_y]$
Igual	07	EQ R_x, R_y	$[R_0] \leftarrow 1$ (se $R_x = R_y$) ou 0 (caso contrário)
Menor	08	LT R_x, R_y	$[R_0] \leftarrow 1$ (se $R_x < R_y$) ou 0 (caso contrário)
Menor ou igual	09	LEQ R_x, R_y	$[R_0] \leftarrow 1$ (se $R_x \leq R_y$) ou 0 (caso contrário)
Ler da memória	10	LDA end	$[R_0] \leftarrow$ [end]
Ler imediato	11	LDI valor	$[R_0] \leftarrow$ valor
Armazenar na memória	12	STR end	[end] $\leftarrow [R_0]$
Copiar registrador	13	MOV R_x, R_y	$[R_x] \leftarrow [R_y]$
Desvio de fluxo	14	JMP end	$[R_{15}] \leftarrow [PC]$ e [PC] $\leftarrow$ end (se $R_0 = 1$)
Pára a computação	15	HLT	Coloca o proc. em modo de espera

```
main() {

  for (int i = 0, i < 10, i++);

}
```

Figura 2: Pequeno programa em C.

```
        LDI 0    //  [R0] <-- 0
        STR 0  //   [0] <-- 0 (assume-se que i esta no endereco 0)
        MOV 1,0  //  [R1] <-- [R0]
        LDI A   //   [R0] <-- 10
        MOV 2,0  //   [R2] <-- [R0]
        LDI 1    //   [R0] <-- 1
        MOV 3,0 //   [R3] <-- [R0]
Ini:    LT 1,2    //   Se [R1] >= [R2], va para Fim
        NOT 0
        JMP Fim
        ADD 3,1 // Caso contrario, [R0] <-- [R3] + [R1]
        MOV 1,0 // [R1] <-- [R0]
        LDI 1   // [R0] <-- 1 (garante o jump)
        JMP ini  // Va para Ini
Fim:    MOV 0,1   // [R0] <-- [R1]
        STR 0  //   i <-- [R0]
        HLT
```

Figura 3: Programa em Assembly da arquitetura ARQ38 correspondente ao programa C da Figura 2.

```
B00 D10 B0A D20 B01 D30 812 200 E0D 331 D10 B01 E06 D01 C00 F00
```

Figura 4: Programa em linguagem de máquina (representada em Hexa) da arquitetura ARQ38 correspondente ao programa C da Figura 2.

A Figura 3 ainda não está na linguagem de máquina, que pode ser obtida substituindo os mneumônicos das instruções pelos seus códigos correspondentes e removendo caracteres especiais, como vírgulas e colchetes, usados na Figura 3 apenas para melhorar a legibilidade. Em linguagem de máquina, o programa dado como exemplo é ilustrado na Figura 4.

3 Regras e especificações para entrega do trabalho

O trabalho é dividido em três fases e deve ser feito por equipes com no máximo dois membros. A partir da entrega da primeira fase, as equipes não poderão trocar ou admitir novos membros. A entrega do trabalho será feita através da página do Moodle associada à disciplina. Cópias de trabalhos, se detectadas, serão desconsideradas. Todos os trabalhos envolvidos na fraude ficarão com nota 0, independente de quem são os verdadeiros autores.

Todo o apoio durante a confecção do projeto será dado pelo professor. Durante as aulas serão reservados intervalos de tempo para retirar possíveis dúvidas e esclarecer qualquer aspecto, tanto relacionado ao projeto, quanto ao uso da ferramenta Logisim. As equipes podem ainda marcar com o professor para discutir aspectos específicos após as aulas, no intervalo entre 15h00 e 16h00.

As fases do trabalho foram concebidas com níveis de dificuldade progressivos e possuem focos em funções específicas da CPU. O cronograma de entrega e o foco de cada fase são descritos a seguir.

Fase 1: 11/11/2018. A implementação da ALU, objetivo nesta fase, é a parte mais simples do trabalho. Apenas circuitos combinacionais são necessários. Muitos destes circuitos estão prontos no Logisim. As equipes precisam configurá-los de forma a atender à especificação do trabalho.

Fase 2: 25/11/2018. Com a ALU já construída, esta deve ser integrada ao Banco de Registradores. Conhecimento sobre a constituição dos registradores estarão sendo vistos em sala nas próximas aulas, subsidiando a execução desta fase. Além dos registradores, multiplexadores e outros circuitos combinacionais fazem parte do REGs. A implementação a ser entregue nesta fase, apesar de ser mais complexa que a ALU, pode ainda ser considerada simples.

Fase 3: 16/12/2018. Nesta fase os demais componentes da arquitetura ARQ38 devem ser construídos e integrados. Esta é a fase que exigirá mais atenção por parte das equipes. Porém, a modelagem da unidade de controle está sendo fornecida neste documento, facilitando a implementação nesta fase do projeto.

A implementação solicitada neste trabalho deverá ser realizada no Logisim, simulador que pode ser obtido através da página `http://www.cburch.com/logisim/`. Esta ferramenta oferece diversas funcionalidades e circuitos integrados já prontos, o que facilita a implementação da arquitetura ARQ38 e sua depuração. Não são impostas quaisquer restrições sobre uso do que já está disponível do Logisim. Ao final do trabalho, a máquina construída no Logisim deve ser capaz de executar um pequeno programa codificado com as instruções da Arquitetura ARQ38. O programa da Figura 4 pode servir como exemplo inicial.

A seguir, informações adicionais relativas a cada fase do trabalho são fornecidas.

3.1 Fase 1: Implementação da ALU

A ALU é apenas um conjunto de circuitos combinacionais capazes de realizar operações lógicas e aritméticas. Na arquitetura ARQ38, a ALU recebe como parâmetros os valores sobre os quais as operações são realizadas, identificados como I_x e I_y na Figura 5, e o código da instrução relativa à operação, no pino Op. Sua única saída é o resultado de tais operações, disponibilizado no pino

Out. Internamente, faz-se uso de multiplexador(es) para selecionar a operação cujo resultado será disponibilizado no pino de saída.

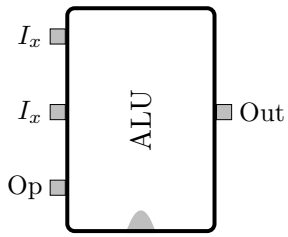

Figura 5: Chip relativo à ALU para a arquitetura ARQ38: três pinos entrada (I_x, I_y e Op) e um de saída (Out).

3.2 Fase 2: Implementação do Banco de Registradores e sua conexão com a ALU

O chip correspondente ao REGs é ilustrado na Figura 6. Há 5 pinos de entrada e 2 de saída. Os três pinos de entrada de 4 bits, Rs1, Rs2 e Rd, são para identificar registradores sobre os quais se quer operar. Se o bit presente no pino de entrada rw_{ctl} for igual a 1, REGs deve colocar o conteúdo dos registradores indicados nos pinos Rs1 e Rs2 nos pinos de saída Rx e Ry, respectivamente. Se $rw_{ctl} = 0$, o conteúdo presente no pino de entrada Rin deve ser gravado no registrador indicado no pino de entrada Rd. Além destes, o bit presente no pino de saída R0 indica 0 ou 1 caso o conteúdo do registrador R_0 seja igual ou diferente de 0, respectivamente. Esta informação é necessária para a implementação de desvio condicional (instrução JMP).

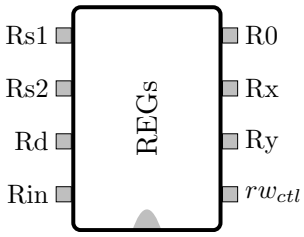

Figura 6: Chip relativo ao REGs para a arquitetura ARQ38: cinco pinos de entrada (Rs1, Rs2, Rd, Rin e rw_{ctl}) e três de saída (R0, Rx e Ry).

3.3 Fase 3: Implementação da CTLU e construção de conexões como os demais componentes

O chip referente à CTLU é ilustrado na Figura 7. Há ao todo seis sinais de controle emitidos pela Unidade de Controle aos demais componentes da CPU e da MD, conforme explicado abaixo. Estes sinais são obtidos após a decodificação da instrução e a verificação do conteúdo de R_0.

- rw_{ctl} (1 bit): Apenas a instrução STR não armazena valores em registradores. Todas as demais colocam em registradores resultados das operações realizadas, o que é indicado pelo sinal de controle $rw_{ctl} = 0$. Em outras palavras, $rw_{ctl} = 1$ apenas quando o código da operação é 12. Para todos os demais, $rw_{ctl} = 0$.
- rin_{ctl} (3 bits): O conteúdo a ser gravado em registradores do REGs pode vir de 5 locais distintos, da ALU (instruções 0-9), da memória (instrução 10), do IR (instrução 11), do próprio REGs (instrução 13) e do PC (instrução 14). O sinal rin_{ctl} indica a fonte que será usada para armazenamento em REGs. Pode-se adotar o seguinte comportamento para o sinal rin_{ctl}, como ilustrado na Tabela 2.

  É interessante notar que o valor de rin_{ctl} para as operações de código 12 e 15 é irrelevante, pois tais operações não alteram valores de registradores. A escolha '000' e '111', como ilustrado na tabela é, portanto, arbitrária.

Tabela 2: Sinal de controle rin_{ctl}.

Código da operação (decimal)	0-9	10	11	12	13	14	15
rin_{ctl} (3 bits)	000	001	010	000	011	100	111

- rd_{ctl} (2 bits): Este sinal indica qual o destino a ser considerado para gravar o conteúdo no REGs. Há 3 possibilidades. A maioria das instruções usa R_0 como registrador destino e a instrução de código 14 guarda o valor atual do PC em R_{15}. Estes dois casos são indicados por $rd_{ctl} = 00$ e $rd_{ctl} = 01$ (em binário), respectivamente. No entanto, para a instrução de código 13, o identificador do registrador destino está indicado na própria instrução, o que é sinalizado em binário por $rd_{ctl} = 10$. Para a instrução de código 15, o sinal é irrelevante. A tabela abaixo adota, neste caso, $rd_{ctl} = 01$.

Tabela 3: Sinal de controle rd_{ctl}.

Código da operação (decimal)	0-11	12-13	14-15
rin_{ctl} (3 bits)	00	10	01

- pc_{ctl} (2 bits). O comportamento usual do PC é ser incrementado após uma instrução ser trazida da memória. Este comportamento é indicado pelo sinal $pc_{ctl} = 01$ (em binário). A instrução de código 14, por outro lado, modifica o conteúdo do PC caso $[R_0] = 1$. Este cenário é sinalizado em binário fazendo $pc_{ctl} = 10$. Por fim, a implementação da instrução de código 15 faz com que PC seja congelado no seu valor corrente, o que é indicado com $pc_{ctl} = 0$. A Tabela 4 resume o comportamento para o sinal pc_{ctl}.

Tabela 4: Sinal de controle pc_{ctl}.

Código da operação (decimal)	0-13	14	14	15
R_0	X	1	$\neq 1$	X
rin_{ctl} (3 bits)	00	10	01	00

  Notar que o valor do sinal pc_{ctl} depende do conteúdo de R_0. De fato, a operação JMP (de código 14), tem comportamentos distintos se o conteúdo de R_0 é igual ou diferente de 1. Em outras palavras, a saída R_0 de REGs deve ser considerada para a correta implementação de pc_{ctl}.

- rm_{ctl} (1 bit) e wm_{ctl} (1 bit). Estes dois sinais de controle referem-se ao comportamento desejado da memória. As operações LDA e STR, quando executadas, realizam leitura e escrita da memória de dados, o que é indicado com $rm_{ctl} = 1$ e $wm_{ctl} = 1$, respectivamente. Para todas as demais instruções, estes sinais de controle estarão em zero. Observe que o comportamento de wm_{ctl} é exatamente o mesmo de rw_{ctl}. Portanto, não é necessário criar estes dois sinais, bastando um deles.

3.4 Outros componentes e algumas dicas

Alguns outros componentes, não listados nas subseções anteriores, são necessários para efetuar as ligações entre os chips REGs, ALU e CTLU. A maioria destes é composto por Mux e/ou Dmux, cujas funções são estabelecer canais de fluxo de informação dentro da CPU. Espera-se que as equipes identifiquem onde usar tais componentes durante o trabalho. Outro componente não detalhado é o PC. Este consiste de um simples contador, cujo comportamento é configurado pelo sinal pc_{ctl}, como

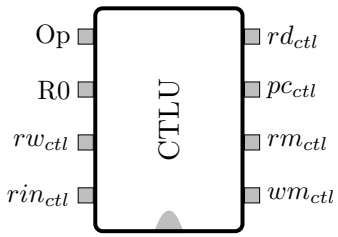

Figura 7: Chip relativo à CTLU para a arquitetura ARQ38: dois pinos de entrada (Op, R0) e seis pinos de saída (rw_{ctl}, rin_{ctl}, rd_{ctl}, pc_{ctl}, rm_{ctl} e wm_{ctl}).

já especificado. As memórias já estão prontas no Logisim. Basta configurá-las de acordo com o especificado para a arquitetura ARQ38.

Para facilitar a visualização e depuração da implementação, recomenda-se o uso intensivo do que é chamado de *túneis* na terminologia Logisim. Estes representam apenas ligações (fios) e são usados para não deixar o projeto poluído com ligações explícitas, o que dificulta tanto o seu entendimento quanto a identificação e a correção de possíveis erros durante a implementação.

No Logisim é possível configurar a largura dos canais de comunicação (em número de bits), agrupando-os em apenas um cabo. É ainda possível instanciar componentes com entrada em saída de largura de bits ajustáveis. Tal funcionalidade ajuda bastante o projeto. Se tal recurso for usado, é necessário em alguns cenários realizar a operação inversa, ou seja, separar os diferentes fios para tratá-los de forma independente. Para tanto, há o componente chamado *distribuidor*. O *extensor de bits* é outro componente que pode ser útil quando se quer estender a largura de um barramento de comunicação.