



**UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
ENGENHARIA MECATRÔNICA
SISTEMAS DIGITAIS**

IMPLEMENTAÇÃO DO PROJETO RTL DE UM MICROCONTROLADOR

NATAL-RN

2020.6

IAGO PEREIRA CASSEL - 20170030770
JARDEL GREGÓRIO DO NASCIMENTO - 20190001829
LIAN GUEDES SILVA - 20190153983
SÂMELA BRUNA FERREIRA - 20200001103

IMPLEMENTAÇÃO DO PROJETO RTL DE UM MICROCONTROLADOR

Relatório apresentado ao componente curricular de Sistemas Digitais do curso de Engenharia Mecatrônica da UFRN como parte da obtenção da nota da 1^a unidade.

Professor: Samaherni Morais Dias

NATAL-RN

2020.6

RESUMO

Foi proposto durante quatro semanas o desenvolvimento da arquitetura de um microcontrolador e implementação do seu projeto RTL utilizando o modelo para simulação computacional. Este relatório dá continuidade e finaliza o projeto de desenvolvimento da arquitetura de um microcontrolador mostrando a etapa da implementação. O projeto RTL do microcontrolador será implementado em VHDL e verificado no Quartus II. Para a verificação desta simulação computacional será desenvolvido códigos fontes em Assembly com base em dois problemas sugeridos pelo roteiro que envolve: SAD e números primos. Também será mostrado os resultados apresentados na simulação que mostrará se a implementação foi viável com base no projeto desenvolvido pelos grupos.

Palavras-chave: microcontrolador, projeto RTL, VHDL, Assembly

LISTA DE FIGURAS

Figura 1 – Esquemático de arquitetura básica de um processador	08
Figura 2 – Diagrama da ULA	12
Figura 3 – Diagrama final do projeto	14
Figura 4 – Decodificador DecodeABC dos registradores de saída	15
Figura 5 – Diagrama detalhado do bloco PC	15
Figura 6 – Diagrama detalhado do banco de registradores RF	16
Figura 7 – Diagrama detalhado de um registrador de 8 bits	16
Figura 8 – Diagrama detalhado do DEMUX de seleção de endereço do banco de registradores	17
Figura 9 – MUX de saída do banco de registradores	18
Figura 10 – Detalhamento de um MUX de saída do banco de registradores	19
Figura 11 – Detalhamento das portas AND habilitadoras de leitura	20
Figura 12 – Diagrama da pilha LIFO	20
Figura 13 – Máquina de Estados	21
Figura 14 – Teste de primalidade	23

LISTA DE TABELAS

Tabela 1 – Conjunto de funções do processador	07
Tabela 2 – Instruções e códigos de operação	09
Tabela 3 – Instruções e códigos da ULA	12
Tabela 4 – Definições do algoritmo da SAD - Parte I	21
Tabela 5 – Definições do algoritmo da SAD - Parte II	22
Tabela 6 – Definições do algoritmo de primalidade	24

SUMÁRIO

1 INTRODUÇÃO	6
2 DESENVOLVIMENTO	7
2.1 Definições de Elementos De Memória e I/O	7
2.2 Codificação de Instruções	7
2.3 Definição de Instruções	8
2.4 Unidade de Lógica e Aritmética (ULA)	11
2.5 Diagramação Final	13
2.6 Máquina de Estados (MDE)	20
2.7 Implementação do projeto RTL em VHDL	21
2.8 Modelos para desenvolver códigos fontes em Assembly	21
2.8.1 SAD	21
2.8.2 Identificador de valor primo	23
3 CONCLUSÃO	25
4 REFERÊNCIAS	26
5 ANEXOS	27

1 INTRODUÇÃO

Nesta semana foi permitido aos grupos a escolha de qualquer outro projeto arquitetado durante as 3 primeiras semanas. Apesar de ter ficado no grupo 5, preferimos implementar o microcontrolador arquitetado pelo grupo 3. Pois acreditamos que ele está de acordo com as especificações solicitadas pelo professor e será possível a implementação de seu projeto RTL em VHDL de forma mais fácil e com poucas alterações do que os outros projetos.

Tabela 1 – Conjunto de funções do processador

Função	Descrição
HLT	parar o processador
MOV	Movimentar dados de 8 bits
ADD	Adicionar 2 dados de 8 bits
SUB	Subtrair 2 dados de 8 bits
CMP	Comparar 2 dados de 8 bits
MUL	Multiplicar 2 dados de 8 bits
INC	Incrementar 1 dado de 8bits
DEC	Decrementar 1 dado de 8bits
AND	Operar a lógica AND entre 2 dados de 8 bits
OR	Operar a lógica OR entre 2 dados de 8 bits
XOR	Operar a lógica XOR entre 2 dados de 8 bits
NOT	Operar a lógica NOT em 1 dado de 8 bits
SHL	Deslocar a esquerda os dados em um conjunto de 8 bits
SHR	Deslocar a direita os dados em um conjunto de 8 bits
JMP	Saltar de forma condicional ou incondicional
PUSH	Incluir dados de 8 bits em uma pilha
POP	Extrair dados de 8 bits de uma pilha
CALL	Implementar a chamada de um segmento do código
RET	Retornar da chamada

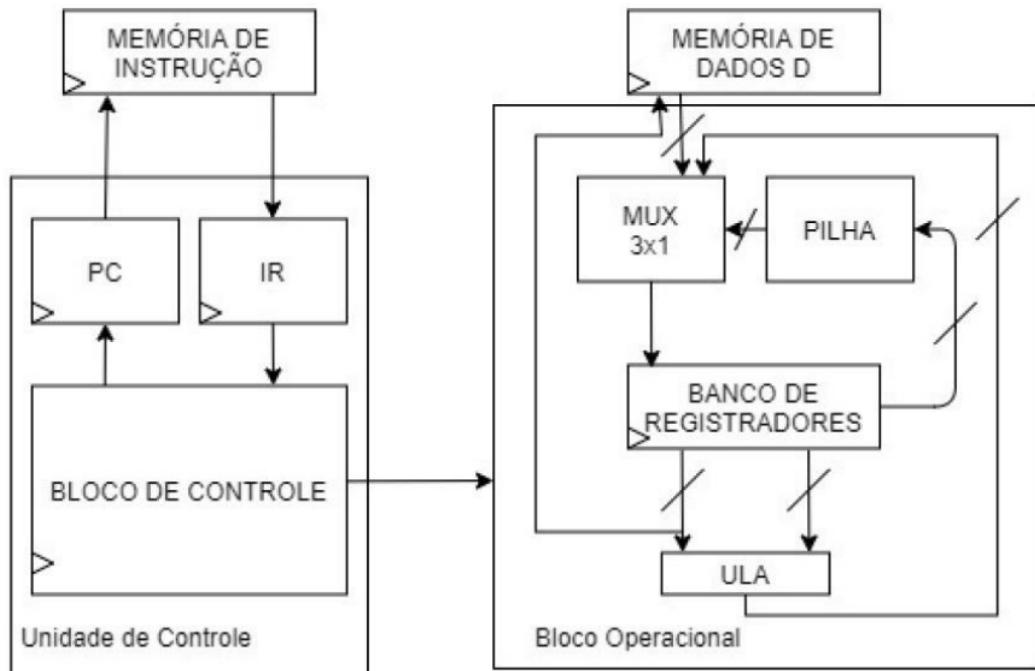
Fonte: Material fornecido pelo professor

2 DESENVOLVIMENTO

2.1 Definições de Elementos De Memória e I/O

- **Memória de instrução:** Armazena as linhas de instrução do programa. Definida como 256 endereços de 17 bits cada.
- **Memória de dados:** Armazena os dados acessíveis do projeto, como dados de entrada e saída. Definida como 256 endereços de 8 bits cada.
- **Pilha:** Serve exclusivamente para as funções PUSH e POP. Definida como 16 endereços de 8 bits.
- **Banco de registradores:** Armazena dados a serem processados pela ULA (Unidade de Lógica e Aritmética) ou dados a serem armazenados na memória de dados. Definida como 16 endereços de 8 bits.
- **Elementos de entrada (D e E):** Foram definidos como vetores externos sendo introduzidos diretamente no MUX prévio ao banco de registradores.
- **Elementos de saída (A, B e C):** Os elementos de saída foram definidos como três registradores de 8 bits, cujos dados são provenientes do banco de registradores RF. O load destes registradores vem de um decodificador.

Figura 1 – Esquemático de arquitetura básica de um processador



Fonte: Elaborado pelo autor

2.2 Codificação de Instruções

No total, foram definidas 26 funções, sendo necessário 5 bits para codificação destas instruções. Essas funções tornam o processador bastante versátil e útil, sendo responsável por solucionar diferentes tipos de problemas que podem serem propostos.

Tabela 2 – Instruções e códigos de operação

Função	Código de operação	Descrição
HLT	00000	Parar o processador
MOVC	00001	Carregar um dado da memória no banco de registradores
MOVA	00010	Armazenar um dado do banco de registradores na memória
ADD	00011	Adicionar 2 dados
SUB	00100	Subtrair 2 dados
CMP	00101	Comparar 2 dados
MUL	00110	Multiplicar 2 dados
INC	00111	Incrementar 1 dado
DEC	01000	Decrementar 1 dado
AND	01001	Operar a lógica AND entre 2 dados
OR	01010	Operar a lógica OR entre 2 dados
XOR	01011	Operar a lógica XOR entre 2 dados
NOT	01100	Operar a lógica NOT em 1 dado
SHL	01101	Deslocar a esquerda os bits de em 1 dado
SHR	01110	Deslocar a direita os bits de em 1 dado
JMP	01111	Saltar de forma condicional ou incondicional
PUSH	10000	Incluir dados em uma pilha
POP	10001	Extrair dados em uma pilha
CALL	10010	Implementar a chamada de um segmento de código
RET	10011	Retornar da chamada
MOVCN	10100	Mover uma constante da instrução para o banco de registradores
JMPEQ	10101	Saltar quando a condição de igual é satisfeita
JMPLO	10110	Saltar quando a condição de menor é satisfeita
JMPHI	10111	Saltar quando a condição de maior é satisfeita
IN	11000	Armazenar dados do I/O no registrador
OUT	11001	Remover dados do I/O no registrador

Fonte: Elaborado pelo autor

2.3 Definição de Instruções

Abaixo temos, de forma resumida, as definições de instruções necessárias para o entendimento dos eventos internos presente no processador.

- **HLT – 00000 xxxx xxxx xxxx:** A máquina de estados para de contar as instruções.

- **MOVC – 00001 $r_3r_2r_1r_0$ d_{7d_{6d_{5d_{4d_{3d_{2d_{1d₀}}}}}: Carrega o dado do endereço d_{7d_{6d_{5d_{4d_{3d_{2d_{1d₀}}}}} da memória de dados no endereço $r_3r_2r_1r_0$ do banco de registradores.}}}}**
- **MOVC – 00001 $r_3r_2r_1r_0$ d_{7d_{6d_{5d_{4d_{3d_{2d_{1d₀}}}}: Carrega o dado do endereço d_{7d_{6d_{5d_{4d_{3d_{2d_{1d₀}}}} da memória de dados no endereço $r_3r_2r_1r_0$ do banco de registradores.}}}}}}**
- **MOVA – 00010 $r_3r_2r_1r_0$ d_{7d_{6d_{5d_{4d_{3d_{2d_{1d₀}}}}: Armazena o dado do endereço $r_3r_2r_1r_0$ do banco de registradores no endereço d_{7d_{6d_{5d_{4d_{3d_{2d_{1d₀}}}} da memória de dados.}}}}}}**
- **ADD – 00011 $ra_3ra_2ra_1ra_0$ rb_{3rb_{2rb_{1rb₀}} rc_{3rc_{2rc_{1rc₀}}: Soma os dados dos endereços rb_{3rb_{2rb_{1rb₀}} e rc_{3rc_{2rc_{1rc₀}} do banco de registradores e armazena no endereço $ra_3ra_2ra_1ra_0$. Exemplo: “00011 0010 0000 0101” especifica RF[2] = RF[0] + RF[5].}}}}**
- **SUB – 00100 $ra_3ra_2ra_1ra_0$ rb_{3rb_{2rb_{1rb₀}} rc_{3rc_{2rc_{1rc₀}}: Subtrai os dados dos endereços rb_{3rb_{2rb_{1rb₀}} e rc_{3rc_{2rc_{1rc₀}} do banco de registradores e armazena no endereço $ra_3ra_2ra_1ra_0$. Exemplo: “00100 0010 0000 0101” especifica RF[2] = RF[0] - RF[5].}}}}**
- **CMP – 00101 $ra_3ra_2ra_1ra_0$ rb<sub>3rb_{2rb_{1rb₀}} rc<sub>3rc_{2rc_{1rc₀}}: Compara os dados dos endereços rb<sub>3rb_{2rb_{1rb₀}} e rc<sub>3rc_{2rc_{1rc₀}} do banco de registradores e armazena o resultado da comparação em $ra_3ra_2ra_1ra_0$. Exemplo: “00101 0010 0000 0101” especifica que RF[2] recebe o resultado da comparação entre RF[0] e RF[5]. São 3 saídas possíveis:

 - o **00000001**: Implica que rb_{3rb_{2rb_{1rb₀}} é menor que rc_{3rc_{2rc_{1rc₀}}.}}
 - o **00000010**: Implica que rb_{3rb_{2rb_{1rb₀}} é maior que rc_{3rc_{2rc_{1rc₀}}.}}
 - o **00000100**: Implica que rb_{3rb_{2rb_{1rb₀}} é igual à rc_{3rc_{2rc_{1rc₀}}.}}</sub></sub></sub></sub>**
- **MUL – 00110 $ra_3ra_2ra_1ra_0$ rb_{3rb_{2rb_{1rb₀}} rc_{3rc_{2rc_{1rc₀}}: Multiplica os dados dos endereços rb_{3rb_{2rb_{1rb₀}} e rc_{3rc_{2rc_{1rc₀}} do banco de registradores e armazena no endereço $ra_3ra_2ra_1ra_0$. Exemplo: “00110 0010 0000 0101” especifica RF[2] = RF[0] * RF[5].}}}}**
- **INC – 00111 $r_3r_2r_1r_0$ xxxx xxxx**: Incrementa o dado no registrador $r_3r_2r_1r_0$ em 1. Exemplo: “00111 0000 xxxx xxxx” especifica RF[0] = RF[0] + 1.
- **DEC – 01000 $r_3r_2r_1r_0$ xxxx xxxx**: Decrementa o dado no registrador $r_3r_2r_1r_0$ em 1. Exemplo: “01000 0000 xxxx xxxx” especifica RF[0] = RF[0] - 1.
- **AND – 01001 $ra_3ra_2ra_1ra_0$ rb_{3rb_{2rb_{1rb₀}} rc_{3rc_{2rc_{1rc₀}}: Realiza a operação lógica AND entre os dados dos endereços rb_{3rb_{2rb_{1rb₀}} e rc_{3rc_{2rc_{1rc₀}} do banco de registradores e armazena o resultado no endereço $ra_3ra_2ra_1ra_0$. Exemplo: “01001 0010 0000 0101” especifica RF[2] = RF[0] AND RF[5].}}}}**
- **OR – 01010 $ra_3ra_2ra_1ra_0$ rb_{3rb_{2rb_{1rb₀}} rc_{3rc_{2rc_{1rc₀}}: Realiza a operação lógica OR entre os dados dos endereços rb_{3rb_{2rb_{1rb₀}} e rc_{3rc_{2rc_{1rc₀}} do banco de registradores e armazena o}}}}**

resultado no endereço $r_3r_2r_1r_0$. Exemplo: “01010 0010 0000 0101” especificando RF[2] = RF[0] OR RF[5].

- **XOR - 01011 $r_3r_2r_1r_0\ rb_3rb_2rb_1rb_0\ rc_3rc_2rc_1rc_0$:** Realiza a operação lógica XOR entre os dados dos endereços $rb_3rb_2rb_1rb_0$ e $rc_3rc_2rc_1rc_0$ do banco de registradores e armazena o resultado no endereço $r_3r_2r_1r_0$. Exemplo: “01011 0010 0000 0101” especificando RF[2] = RF[0] XOR RF[5].

- **NOT - 01100 $r_3r_2r_1r_0\ rb_3rb_2rb_1rb_0\ xxxx$:** Realiza a operação lógica NOT no dado do endereço $rb_3rb_2rb_1rb_0$ do banco de registradores e armazena o resultado no endereço $r_3r_2r_1r_0$. Exemplo: “01100 0010 0000 xxxx” especificando RF[2] = NOT RF[0].

- **SHL - 01101 $r_3r_2r_1r_0\ xxxx\ xxxx$:** Realiza o deslocamento de bits para esquerda no dado do endereço $r_3r_2r_1r_0$ do banco de registradores. Exemplo: “01101 0010 xxxx xxxx” especificando RF[2] = $<<$ RF[2]. Nota-se que esta operação pode ser feita através de uma multiplicação por 2 ou manipulando individualmente os bits do dado especificado.

- **SHR - 01110 $r_3r_2r_1r_0\ xxxx\ xxxx$:** Realiza o deslocamento de bits para direita no dado do endereço $r_3r_2r_1r_0$ do banco de registradores. Exemplo: “01110 0010 xxxx xxxx” especificando RF[2] = $>>$ RF[2]. Nota-se que esta operação pode ser feita através de uma divisão por 2 ou manipulando individualmente os bits do dado especificado.

- **JMP – 01111 $xxxx\ d_7d_6d_5d_4d_3d_2d_1d_0$:** Realiza um salto nas instruções por alterar o valor do PC para o valor atual do PC + $d_7d_6d_5d_4d_3d_2d_1d_0$, é importante subtrair o resultado da soma por 1 já que o PC já havia sido incrementado previamente. D_7 é um bit de sinal.

- **PUSH – 10000 $r_3r_2r_1r_0\ xxxx\ xxxx$:** Armazena o dado do endereço $r_3r_2r_1r_0$ do banco de registradores na pilha. Exemplo: “10000 0001 xxxx xxxx” especificando RF[1] → PI.

- **POP – 10001 $r_3r_2r_1r_0\ xxxx\ xxxx$:** Armazena no endereço $r_3r_2r_1r_0$ do banco de registradores o último valor registrado na pilha. Exemplo: “10000 0001 xxxx xxxx” especificando PI → RF[1].

- **CALL – 10010 $xxxx\ d_7d_6d_5d_4d_3d_2d_1d_0$:** Chama uma tarefa especificada no endereço $d_7d_6d_5d_4d_3d_2d_1d_0$ e o armazena para que seja possível executar a instrução seguinte após a finalização da atual.

- **RET – 10011:** Finaliza a tarefa e retorna para a instrução seguinte àquela que chamou a tarefa.

- **MOVCN - 10100 $r_3r_2r_1r_0\ d_7d_6d_5d_4d_3d_2d_1d_0$:** Guaude no endereço $r_3r_2r_1r_0$ do banco de registradores o valor da constate $d_7d_6d_5d_4d_3d_2d_1d_0$. Ela move o valor de uma constante desejada, vinda na instrução, diretamente para o banco de registradores.

- **JMPEQ – 10101 $r_3r_2r_1r_0 d_7d_6d_5d_4d_3d_2d_1d_0$:** Compara se o valor do endereço $r_3r_2r_1r_0$ do banco de registradores é igual a um registrador de 8 bits preestabelecido, se satisfeita ela pula as instruções $d_7d_6d_5d_4d_3d_2d_1d_0$.

- **JMPLO – 10110 $r_3r_2r_1r_0 d_7d_6d_5d_4d_3d_2d_1d_0$:** compara se o valor do endereço $r_3r_2r_1r_0$ do banco de registradores é menor do que um registrador de 8 bits preestabelecido, se satisfeita ela pula as instruções $d_7d_6d_5d_4d_3d_2d_1d_0$.

- **JMPHI – 10111 $r_3r_2r_1r_0 d_7d_6d_5d_4d_3d_2d_1d_0$:** Compara se o valor do endereço $r_3r_2r_1r_0$ do banco de registradores é maior do que um registrador de 8 bits preestabelecido, se satisfeita ela pula as instruções $d_7d_6d_5d_4d_3d_2d_1d_0$.

- **IN - 11000 $r_3r_2r_1r_0 xxxx xxdd_0$:** Coloque dentro do registrador $r_3r_2r_1r_0$ o que está na entrada d_0 . Como temos apenas duas entradas no projeto (D e E), quando d_0 for 0 irá representar D e quando d_0 for 1 irá representar E.

- **OUT - 11001 $r_3r_2r_1r_0 xxxx xxdd_0$:** Remova do registrador $r_3r_2r_1r_0$ o que está na entrada d_1d_0 . Como temos três saídas no projeto (A, B e C), quando d_1d_0 for 01 irá representar A, quando d_1d_0 for 10 irá representar B e quando d_1d_0 for 11 irá representar C e 00 não há saídas a serem escritas.

2.4 Unidade de Lógica e Aritmética (ULA)

A ULA deste projeto foi encarregada de 12 operações e possui duas entradas. Foi proposto que as duas entradas entrem em todas as operações ao mesmo tempo (a não ser as operações que só precisem de uma entrada do banco de registradores, como INC, DEC, NOT, SHL e SHR). Sendo necessário um vetor de seleção de operação de 4 bits (0000 a 1011) para compreender todas as 12 operações desta unidade.

Tabela 3 – Instruções e códigos da ULA

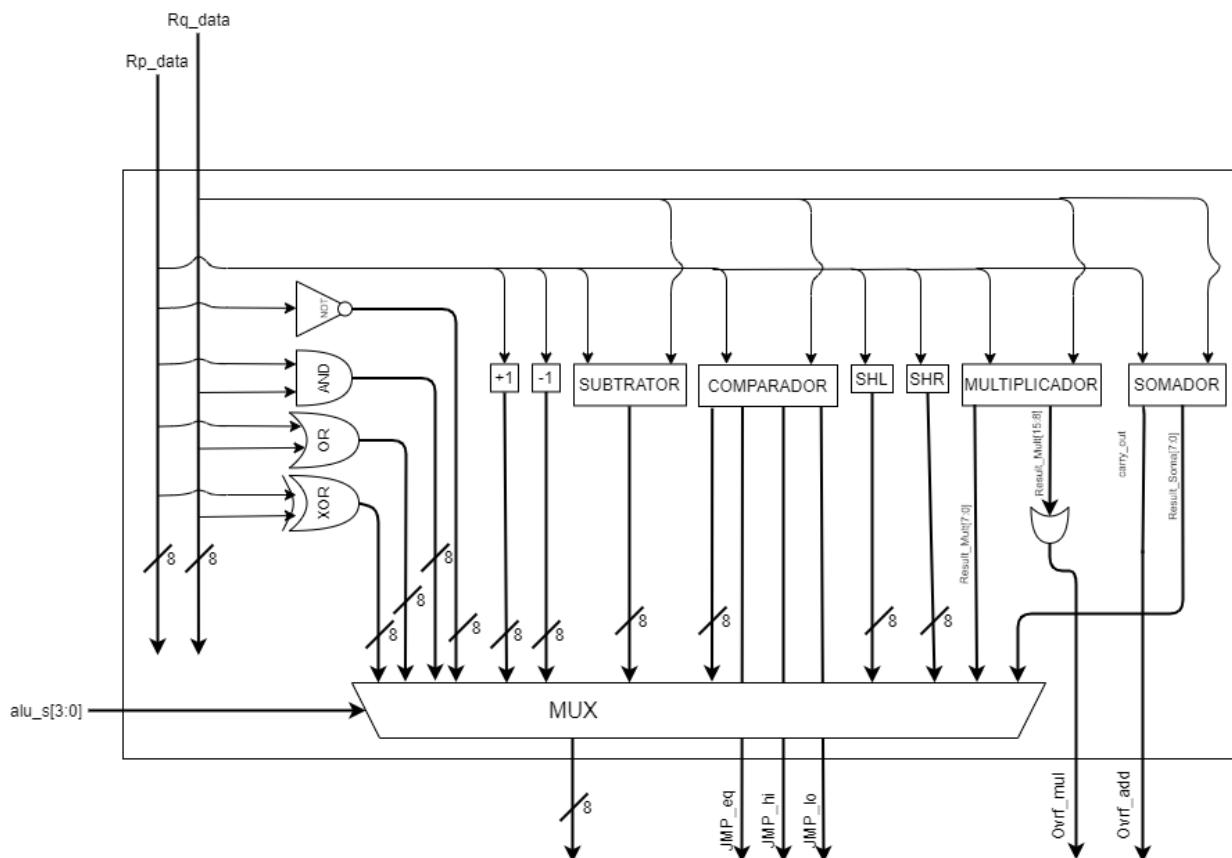
Código de seleção de operação	Operação
0000	ADD
0001	SUB
0010	CMP
0011	MUL
0100	INC
0101	DEC
0110	AND
0111	OR
1000	XOR
1001	NOT
1010	SHL
1011	SHR

Fonte: Elaborado pelo autor

Há ainda flags para realizar comparações nos registradores e sinalizar quando ocorrer overflow. Os flags de comparação vêm do próprio bloco de comparação previamente estabelecido da ULA, sem necessidade de adição de blocos. Um exemplo da utilização do flag de overflow, seria a sinalização da entre multiplicação de dois registradores de 8 bits com valores altos, onde o resultado ultrapassa os 8 bits causando um overflow internamente na ULA.

As flags de overflow agora são ativadas por portas lógicas OR, e não mais com o comparador (maior do que zero). É verificado os oito bits mais significativos do vetor de 16 bits de saída do multiplicador, para saber se algum bit é 1, caso positivo, existe overflow e a saída do carry irá direto para a saída do overflow. Antes as saídas carry do somador e do multiplicador entravam em uma porta lógica OR, cuja saída era a sinalização de overflow. No entanto foi decidido remover a porta lógica OR e ter duas saídas de overflow uma para adição e outra para a multiplicação, para dessa forma ficar mais fácil diferenciar as flags.

Figura 2 – Diagrama da ULA

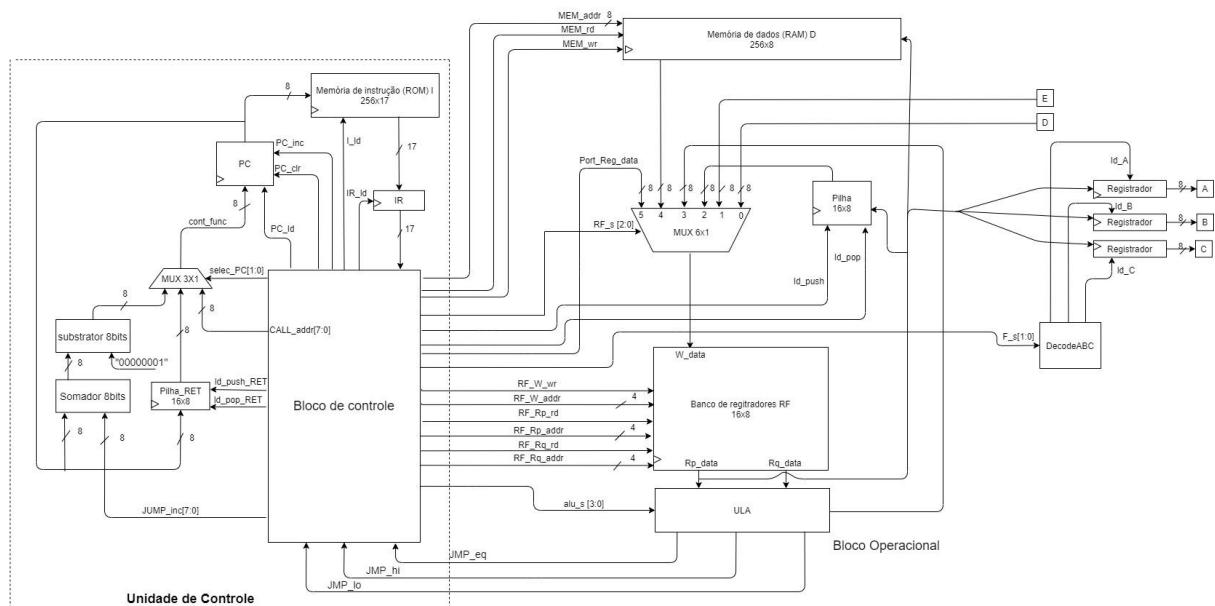


Fonte: Elaborado pelo autor

2.5 Diagramação Final

A seguir será apresentado a diagramação final do projeto realizado durante as 3 primeiras semanas, além de algumas leves alterações realizada pelo grupo atual para facilitar a implementação em VHDL. Veremos a diagramação da Unidade de Controle e do Bloco Operacional, bem como a diagramação detalhada de outros componentes essenciais ao projeto:

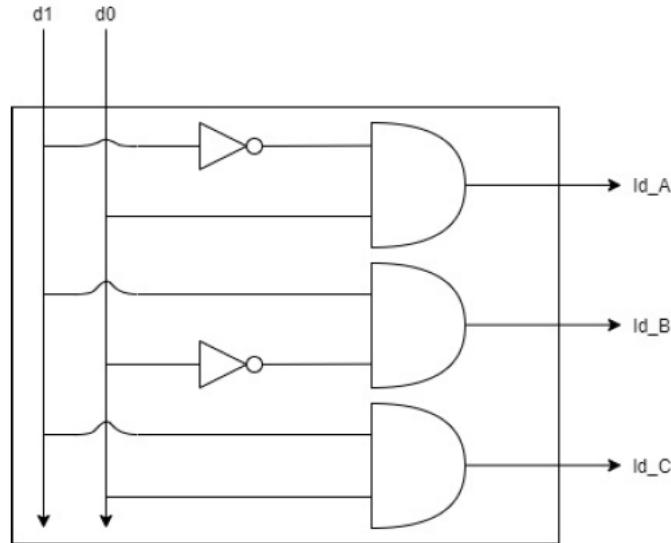
Figura 3 – Diagrama final do projeto



Fonte: Elaborado pelo autor

Durante a implementação deste projeto tivemos a necessidade de alterar o mux que fica em cima do banco de registradores. Uma pequena mudança foi realizada na sua entrada ficando: a entrada E na entrada 1 do mux e a entrada D na entrada 0 do mux. Essa mudança foi feita para tornar a troca de informações mais convenientes e facilitar a transição de dados do circuito.

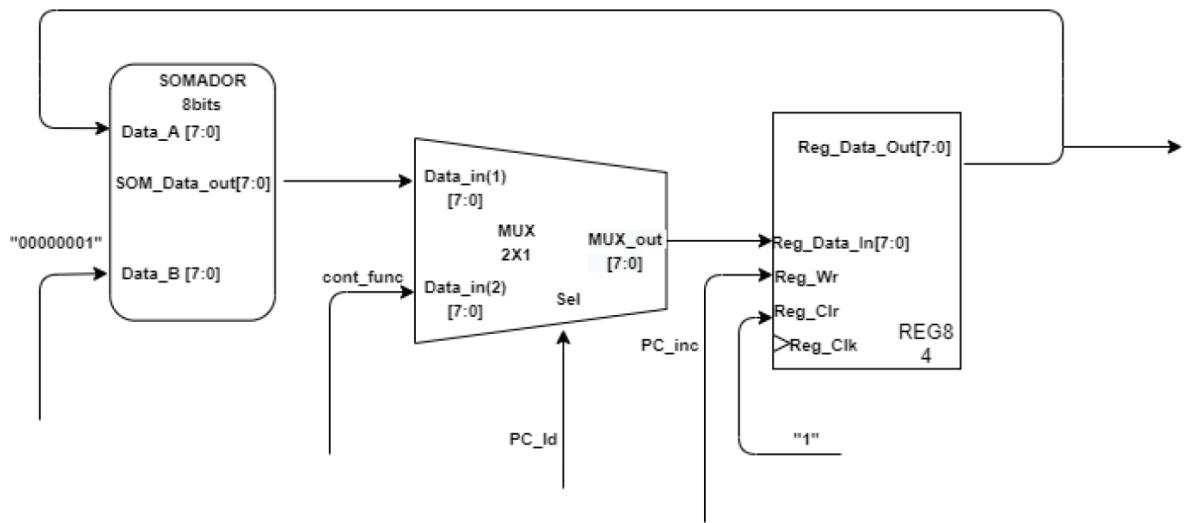
Figura 4 – Decodificador DecodeABC dos registradores de saída



Fonte: Elaborado pelo autor

O bloco PC pode receber três valores provenientes de um MUX externo ao bloco. Um dos valores é o endereço desejado da instrução CALL. Outro valor é o endereço atual para retorno da função RET. Outro valor é referente ao salto da instrução JUMP. Vale ressaltar que, dentro do próprio bloco, há um valor autoincrementável que representa o percurso natural do contador. Outro MUX, dessa vez interno ao bloco, decide entre o valor do MUX externo e o valor corrente do contador através do bit de seleção PC_Id.

Figura 5 – Diagrama detalhado do bloco PC

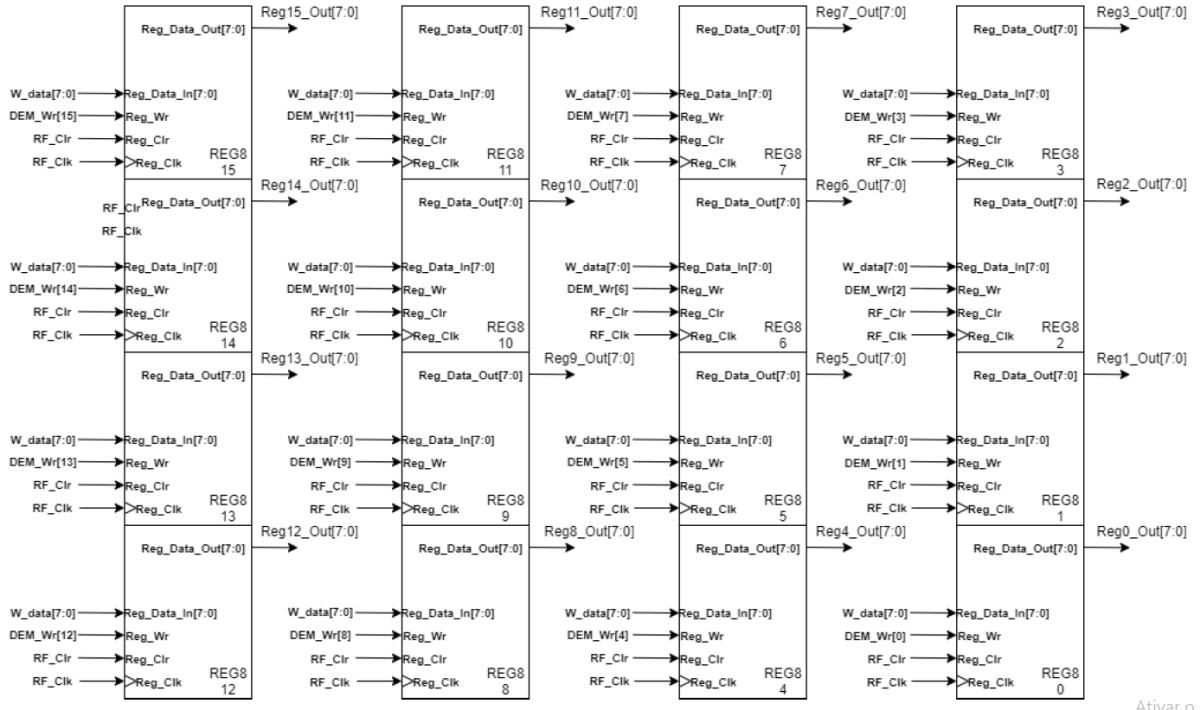


Fonte: Elaborado pelo autor

Ativar ↴

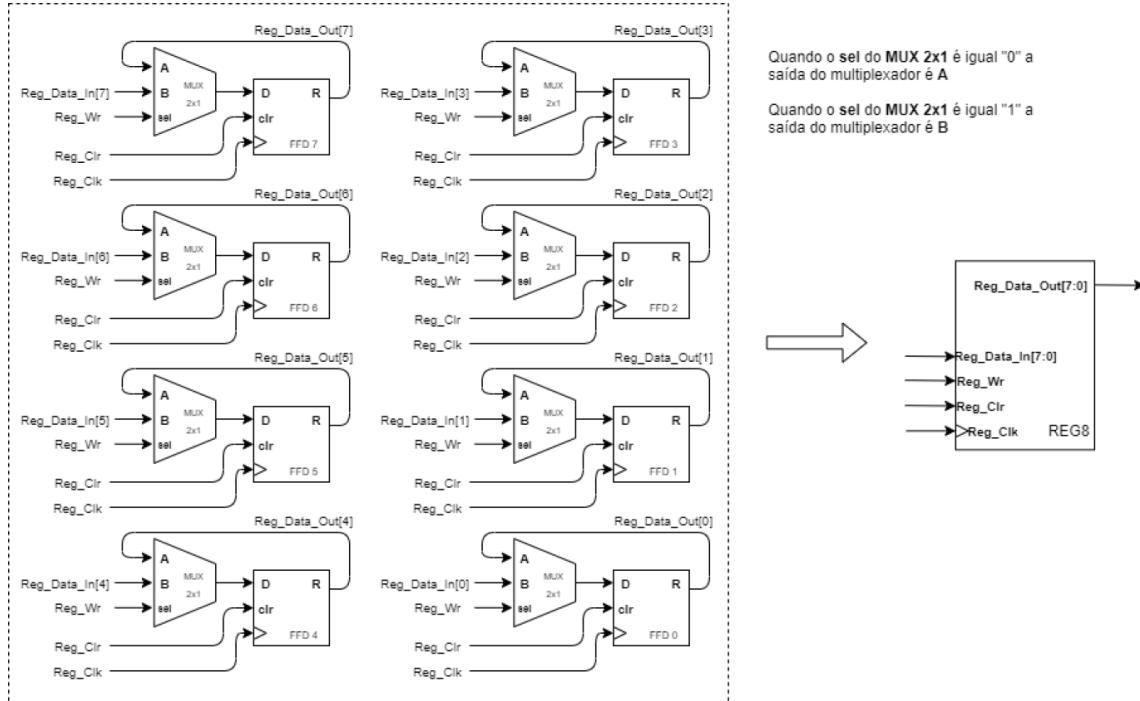
Por convenção, foi explicitado a entrada de clear dos registradores como RF_Clr, porém ela não foi utilizada na implementação do projeto.

Figura 6 – Diagrama detalhado do banco de registradores RF



Fonte: Elaborado pelo autor

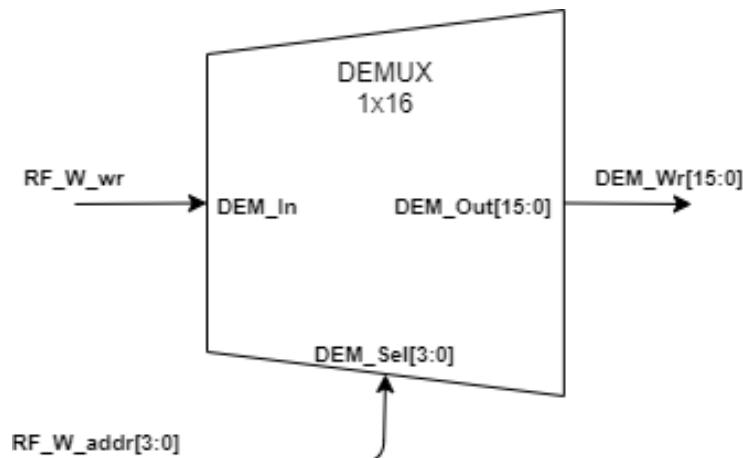
Figura 7 – Diagrama detalhado de um registrador de 8 bits



Fonte: Elaborado pelo autor

Vale ressaltar que a escrita no registrador selecionado do banco RF será ativada por um demultiplexador de uma entrada e dezesseis saídas, conforme a figura abaixo:

Figura 8 – Diagrama detalhado do DEMUX de seleção de endereço do banco de registradores

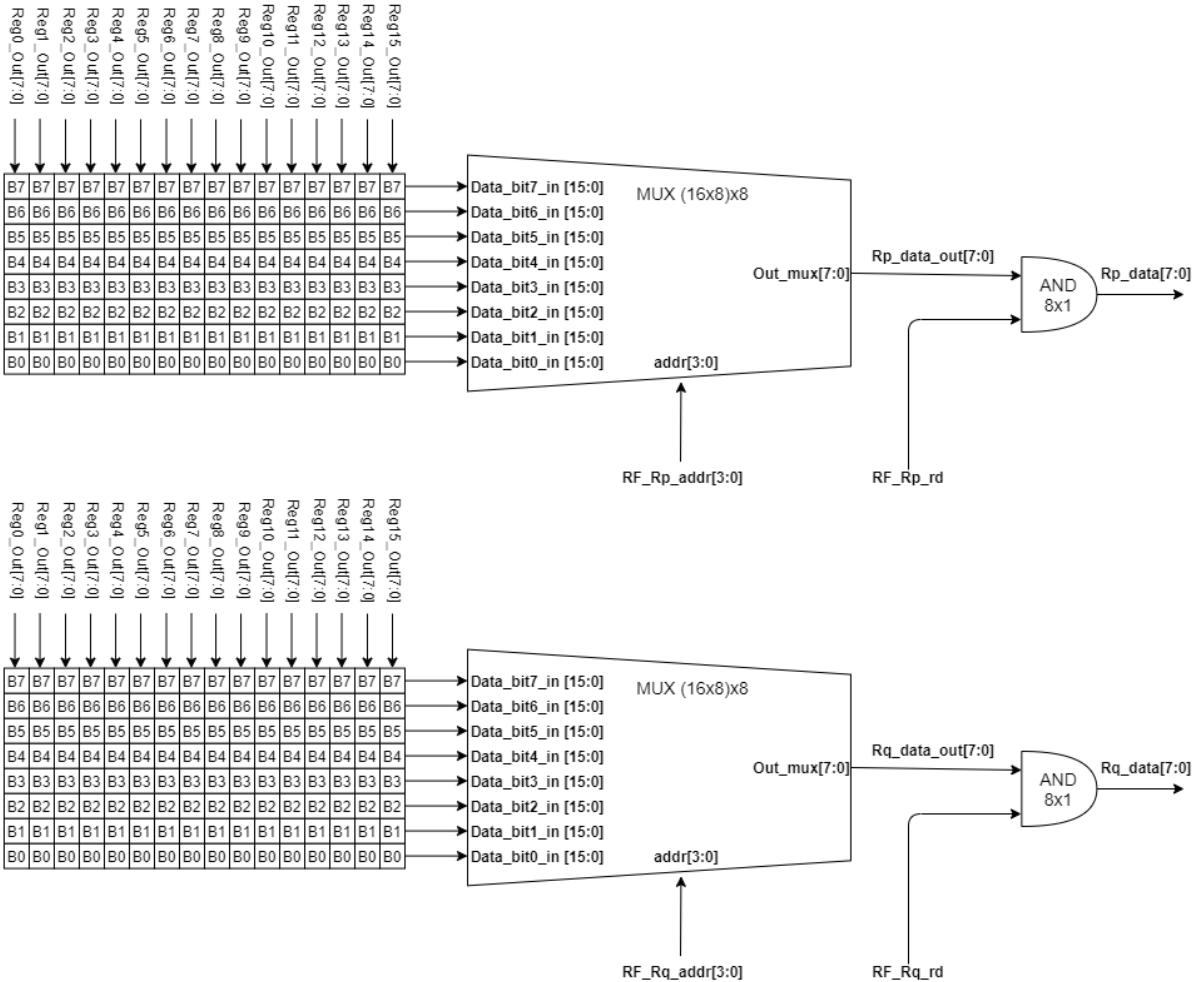


Fonte: Elaborado pelo autor

Todos os registradores entram em dois multiplexadores diferentes, cujas saídas são Rp_data e Rq_data. Estes multiplexadores funcionam de forma estruturada aos vetores de entrada, de modo que possuem oito entradas de dezesseis bits. Cada uma dessas entradas receberá o bit relativo a uma posição de todos os 16 registradores. Desta forma, o vetor de seleção definirá qual é a sequência de bits correta relativa ao registrador desejado. Por exemplo:

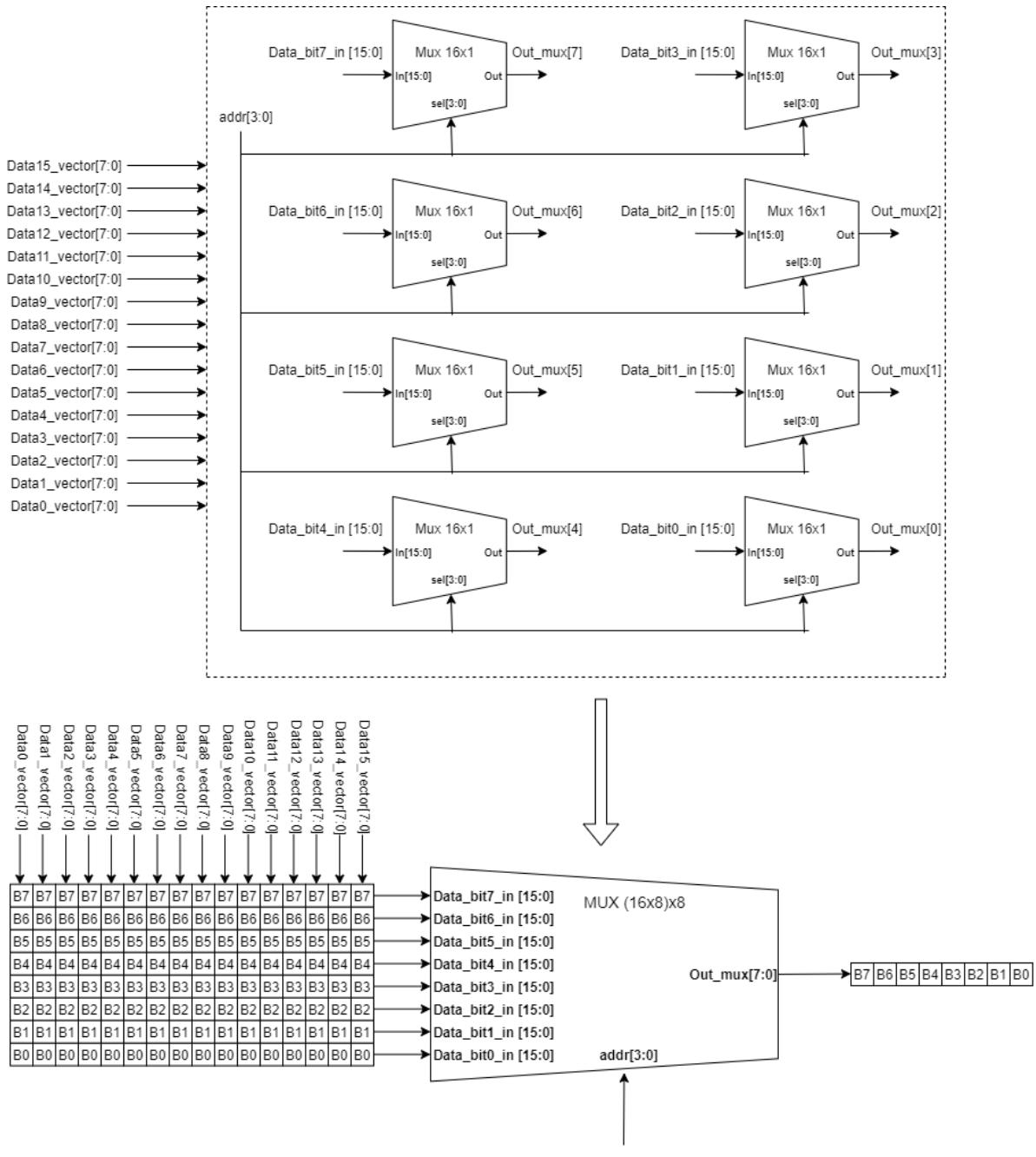
Em Data_bit7_in, entram todos os dezesseis bits referentes a posição [7] dos registradores, em Data_bit6_in, entram todos os dezesseis bits referentes a posição [6] dos registradores, e assim por diante. Se quisermos o vetor de seleção addr for “0010”, estaremos escolhendo o bit na posição [2] do vetor Data_bit7_in, o bit na posição [2] do vetor Data_bit6_in, e assim por diante, constituindo na saída o vetor de oito bits referente ao registrador de posição [2].

Figura 9 – MUX de saída do banco de registradores



Fonte: Elaborado pelo autor

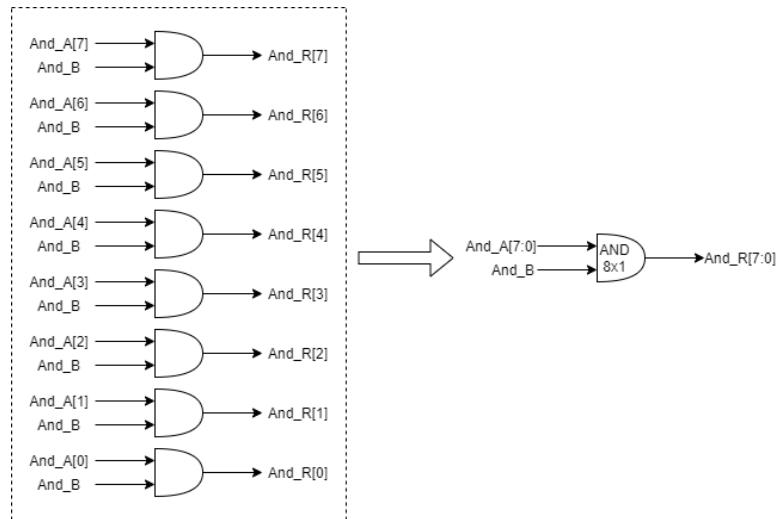
Figura 10 – Detalhamento de um MUX de saída do banco de registradores



Fonte: Elaborado pelo autor

Há ainda uma porta AND 8x1 na saída destes multiplexadores a fim de habilitar a leitura.

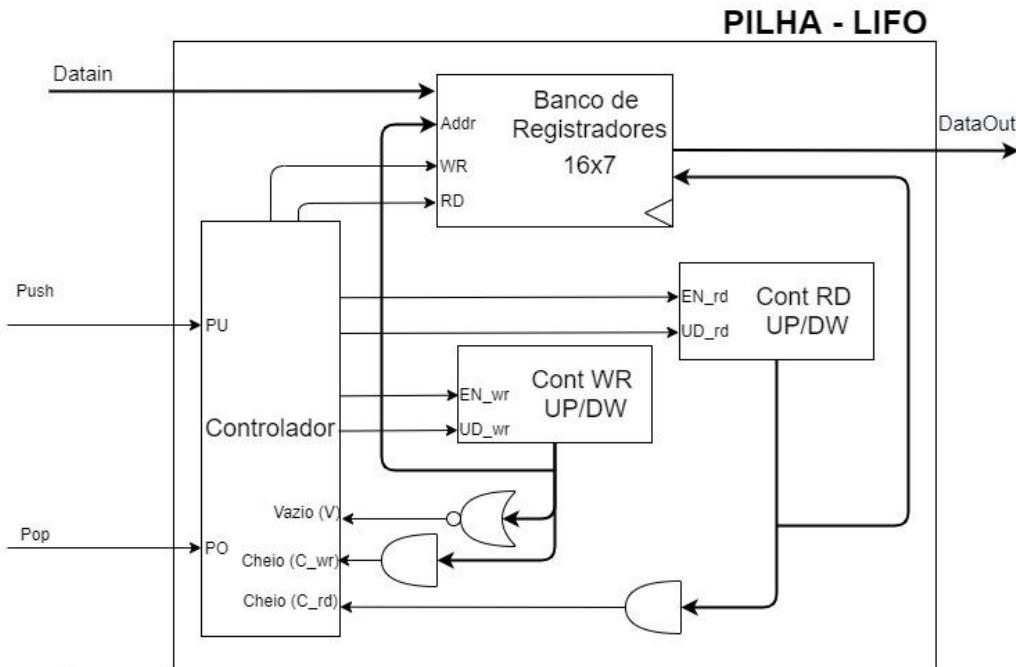
Figura 11 – Detalhamento das portas AND habilitadoras de leitura



Fonte: Elaborado pelo autor

A organização da pilha consta conforme o diagrama abaixo. Vale ressaltar que o bit de seleção do MUX que decide entre o somador e o subtrator deve vir do bloco de controle e condizer com a instrução de POP ou PUSH.

Figura 12 – Diagrama da pilha LIFO



Lógicas do controlador:

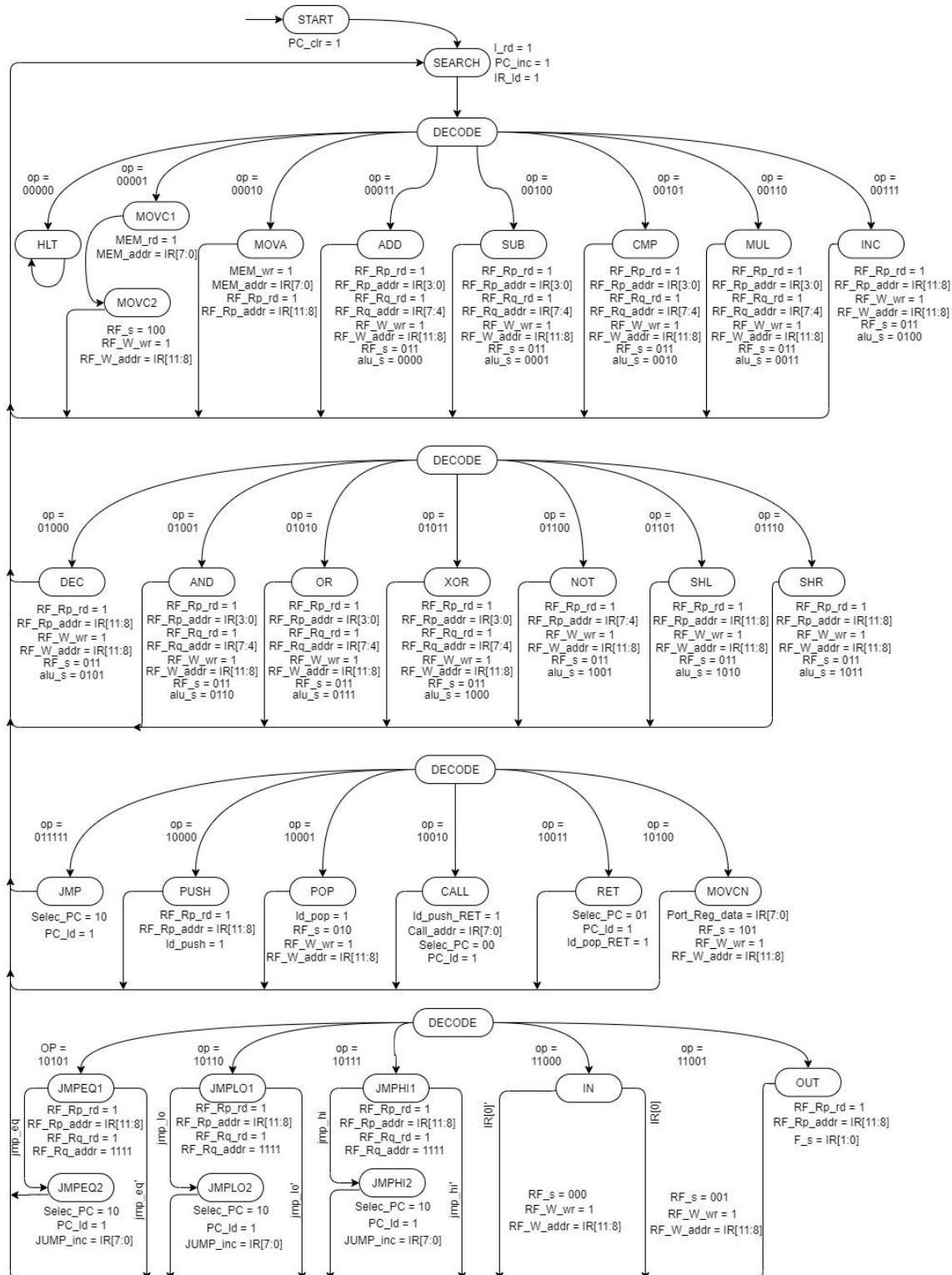
$$\begin{aligned}
 UD_{wr} &= (C_{wr'} * C_{rd'} * PU * PO') \text{ OR } (C_{wr'} * V * PU * PO') \\
 EN_{wr} &= (C_{wr'} * V' * PU * PO) \text{ OR } (V' * C_{rd'} * PU * PO) \text{ OR } (C_{wr'} * C_{rd'} * PU * PO') \text{ OR } (C_{wr'} * V * PU * PO') \\
 UR_{rd} &= (V' * C_{rd'} * PU * PO') \text{ OR } (C_{wr'} * C_{rd} * PU * PO) \\
 EN_{rd} &= (V' * C_{rd} * PU * PO') \text{ OR } (V' * PU * PO) \text{ OR } (C_{wr'} * C_{rd} * PU * PO') \\
 WR &= (V' * C_{rd'} * PU * PO') \text{ OR } (C_{wr'} * PU * PO') \\
 RD &= '1'
 \end{aligned}$$

Fonte: Elaborado pelo autor

2.6 Máquina de Estados (MDE)

No modelo da máquina de estados teremos os estados de “START” para iniciar o processador e a contagem de instruções, “SEARCH” que irá capturar a instrução de cada sequência e “DECODE” que recebe o opcode e direciona para o estado desejado.

Figura 13 – Máquina de Estados



Fonte: Elaborado pelo autor

2.7 Implementação do projeto RTL em VHDL

Com base no projeto RTL conseguimos elaborar um modelo do microcontrolador em VHDL, com isso conseguimos simular como seria o funcionamento real das funções presentes nesse processador. No Anexos será mostrado cada componente e bloco que foi desenvolvido para realizar a simulação computacional em VHDL.

2.8 Modelos para desenvolver códigos fontes em Assembly

Para a verificação do funcionamento do processador atendendo todos os requisitos que são pedidos no projeto, foi necessário o desenvolvimento de códigos fontes em Assembly que exige que algumas funcionalidades do microcontrolador sejam executadas no processo de simulação. No roteiro do projeto foi sugerido dois tipos de problemas sendo eles: a implementação de uma SAD entre dois vetores de 10 posições e um código para identificar se um número dado pelo usuário é primo ou não.

2.8.1 SAD

Para a resolução do primeiro problema do roteiro implementamos um código fonte em Assembly para achar a soma da diferença absoluta (SAD) entre dois vetores de 10 posições. A SAD é basicamente uma métrica de similaridade para decisão de modo de codificação, representando a maior parte da complexidade computacional dos codificadores de vídeos.

Foi implementado um modelo utilizando as instruções para solucionar esse problema. O primeiro vetor de dados que será subtraído está guardado na posição 0 até a 9 e o segundo vetor está guardado na posição 10 até a 19 na memória de dados. A resposta final será apresentada no registrador A do I/O.

Tabela 4 – Definições do algoritmo da SAD - Parte I

Nº da instrução	Instrução	OPCODE				Descrição	
0	00000000	MOVCN	10100	0011	00000000	Move o valor 0 para o 4º reg.	
1	00000001	MOVC	00001	0000	00000000	Carrega o dado da 1ª posição da memória no 1º reg. do banco	
2	00000010	MOVC	00001	0001	00001010	Carrega o dado da 11ª posição da memória no 2º reg. do banco	
3	00000011	CALL DIF_ABS	10010		101010	Calcula o valor da diferença absoluta entre dois dados e armazena no 10º reg. do banco	Calculo da 1ª diferença
4	00000100	ADD	00011	0011	0011 1001	Coloca a soma do 4º e 10º reg. no 4º registrador	
5	00000101	MOVC	00001	0000	00000001	Carrega o dado da 2ª posição da memória no 1º reg. do banco	
6	00000110	MOVC	00001	0001	00001011	Carrega o dado da 12ª posição da memória no 2º reg. do banco	
7	00000111	CALL DIF_ABS	10010		101010	Calcula o valor da diferença absoluta entre dois dados e armazena no 10º reg. do banco	Calculo da 2ª diferença
8	00001000	ADD	00011	0011	0011 1001	Coloca a soma do 4º e 10º reg. no 4º registrador	
9	00001001	MOVC	00001	0000	00000010	Carrega o dado da 3ª posição da memória no 1º reg. do banco	
10	00001010	MOVC	00001	0001	00001100	Carrega o dado da 13ª posição da memória no 2º reg. do banco	
11	00001011	CALL DIF_ABS	10010		101010	Calcula o valor da diferença absoluta entre dois dados e armazena no 10º reg. do banco	Calculo da 3ª diferença
12	00001100	ADD	00011	0011	0011 1001	Coloca a soma do 4º e 10º reg. no 4º registrador	

Fonte: Elaborado pelo autor

Tabela 5 – Definições do algoritmo da SAD - Parte II

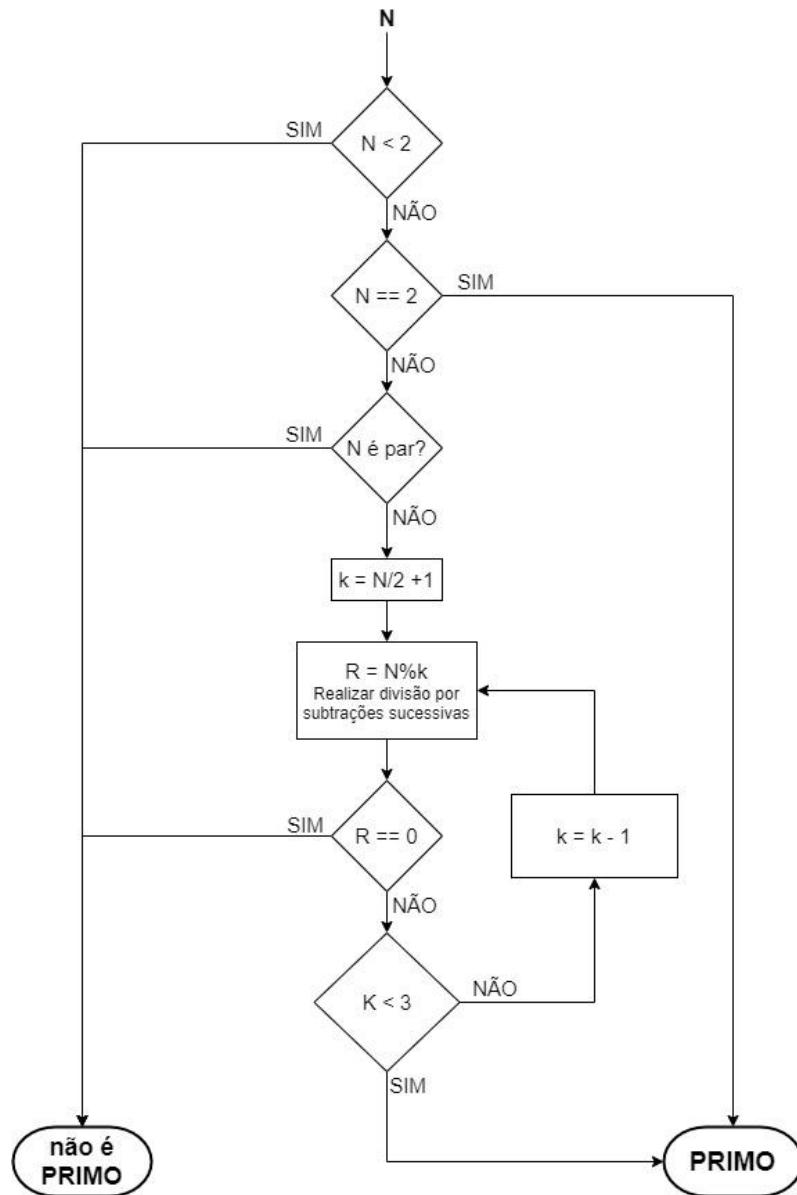
13	00001101	MOVC	00001	0000	00000011		Carrega o dado da 4ª posição da memória no 1º reg. do banco	Calculo da 4ª diferença
14	00001110	MOVC	00001	0001	00001101		Carrega o dado da 14ª posição da memória no 2º reg. do banco	
15	00001111	CALL DIF_ABS	10010		101010		Calcula o valor da diferença absoluta entre dois dados e armazena no 10º reg. do banco	
16	00010000	ADD	00011	0011	0011	1001	Coloca a soma do 4º e 10º reg. no 4º registrador	
17	00010001	MOVC	00001	0000	00000100		Carrega o dado da 5ª posição da memória no 1º reg. do banco	Calculo da 5ª diferença
18	00010010	MOVC	00001	0001	00001110		Carrega o dado da 15ª posição da memória no 2º reg. do banco	
19	00010011	CALL DIF_ABS	10010		101010		Calcula o valor da diferença absoluta entre dois dados e armazena no 10º reg. do banco	
20	00010100	ADD	00011	0011	0011	1001	Coloca a soma do 4º e 10º reg. no 4º registrador	
21	00010101	MOVC	00001	0000	00000101		Carrega o dado da 6ª posição da memória no 1º reg. do banco	Calculo da 6ª diferença
22	00010110	MOVC	00001	0001	00001111		Carrega o dado da 16ª posição da memória no 2º reg. do banco	
23	00010111	CALL DIF_ABS	10010		101010		Calcula o valor da diferença absoluta entre dois dados e armazena no 10º reg. do banco	
24	00011000	ADD	00011	0011	0011	1001	Coloca a soma do 4º e 10º reg. no 4º registrador	
25	00011001	MOVC	00001	0000	00000110		Carrega o dado da 7ª posição da memória no 1º reg. do banco	Calculo da 7ª diferença
26	00011010	MOVC	00001	0001	00010000		Carrega o dado da 17ª posição da memória no 2º reg. do banco	
27	00011011	CALL DIF_ABS	10010		101010		Calcula o valor da diferença absoluta entre dois dados e armazena no 10º reg. do banco	
28	00011100	ADD	00011	0011	0011	1001	Coloca a soma do 4º e 10º reg. no 4º registrador	
29	00011101	MOVC	00001	0000	00000111		Carrega o dado da 8ª posição da memória no 1º reg. do banco	Calculo da 8ª diferença
30	00011110	MOVC	00001	0001	00010001		Carrega o dado da 18ª posição da memória no 2º reg. do banco	
31	00011111	CALL DIF_ABS	10010		101010		Calcula o valor da diferença absoluta entre dois dados e armazena no 10º reg. do banco	
32	00100000	ADD	00011	0011	0011	1001	Coloca a soma do 4º e 10º reg. no 4º registrador	
33	00100001	MOVC	00001	0000	000001000		Carrega o dado da 9ª posição da memória no 1º reg. do banco	Calculo da 9ª diferença
34	00100010	MOVC	00001	0001	00010010		Carrega o dado da 19ª posição da memória no 2º reg. do banco	
35	00100011	CALL DIF_ABS	10010		101010		Calcula o valor da diferença absoluta entre dois dados e armazena no 10º reg. do banco	
36	00100100	ADD	00011	0011	0011	1001	Coloca a soma do 4º e 10º reg. no 4º registrador	
37	00100101	MOVC	00001	0000	000001001		Carrega o dado da 10ª posição da memória no 1º reg. do banco	Calculo da 10ª diferença
38	00100110	MOVC	00001	0001	00010011		Carrega o dado da 20ª posição da memória no 2º reg. do banco	
39	00100111	CALL DIF_ABS	10010		101010		Calcula o valor da diferença absoluta entre dois dados e armazena no 10º reg. do banco	
40	00101000	ADD	00011	0011	0011	1001	Coloca a soma do 4º e 10º reg. no 4º registrador	
41	00101001	OUT	11001	0011	000000001		Envia o dado so 4º reg. do banco para o registrador A do I/O	
42	00101010	MOVCN	10100	1110	00000000		Move o valor 0 para o 15º reg.	DIFERENÇA ABSOLUTA
43	00101011	ADD	00011	1111	0000	1110	Coloca a soma do 1º e 15º reg. no 16º registrador	
44	00101100	JMPHI	10110	0001	00000011		Se o valor do 2º reg. for maior que o do 1º reg. é realizado um pulo de 3 instruções	
45	00101101	SUB	00100	1001	0000	0001	Realiza a subtração entre os dados do 1º e do 2º reg. e armazena o resultado no 10º reg. (sub A - B)	
46	00101110	RET	10011				Realiza a subtração entre os dados do 2º e do 1º reg. e armazena o resultado no 10º reg. (sub B - A)	
47	00101111	SUB	00100	1001	0001	0000		
48	00110000	RET	10011					

Fonte: Elaborado pelo autor

2.8.2 Identificador de valor primo

O segundo problema exige a implementação de um código fonte em Assembly para identificar se um número dado pelo usuário é um número primo ou não é. Para verificação do resultado é proposto que se o valor for primo, deve ser escrito o valor 49 na posição de memória 232, do contrário o valor a ser escrito será 48.

Figura 14 – Teste de primalidade



Fonte: Elaborado pelo autor

O teste da primalidade é um modelo utilizado em algoritmos com a finalidade de determinar se o valor é primo. Com base nesse modelo foi elaborado um código em Assembly utilizando as instruções das funções do microcontrolador. O resultado é apresentado na próxima tabela:

Tabela 6 – Definições do algoritmo de primalidade

Nº da instrução	Instrução	OPCODE				Descrição	
0	00000000	IN	11000	0000	00000001	Coloca o valor do registrador E do I/O no primeiro registrador do Banco	ALGORITMO DE PRIMALIDADE
1	00000001	MOVCN	10100	1111	00000010	Coloca o valor 2 no 16º registrador do Banco	
2	00000010	JMPLO	10110	0000	00100001	Se o valor do 1º reg. for menor que o do 16º reg. é realizado um pulo de 33 instruções (não é primo)	
3	00000011	JMPEQ	10101	0000	00011110	Se o valor do 1º reg. for igual que o do 16º reg. é realizado um pulo de 30 instruções (é primo)	
4	00000100	CALL (PAR?)	10010	xxxx	00001111	Chamada do procedimento que testa se o valor é par (pula para instrução 15)	
5	00000101	ADD	00011	0101	0000	Soma os dados do 1º e 16º reg. e guarda no 6º reg.	
6	00000110	SHR	01110	0101		Desloca o valor do 6º reg. Para a direita, o que equivale a dividir por 2	
7	00000111	INC	00111	0101		Incrementa o valor do 6º reg.	
8	00001000	CALL (DIVISÃO)	10010		00011010	Chamada do procedimento que calcula a divisão por meio de sucessivas subtrações (pula para instrução 26)	
9	00001001	MOVCN	10100	1111	00000000	Move o valor 0 para o 16º reg.	
10	00001010	JMPEQ	10101	0110	00011001	Se o valor dos 7º e 16º reg. forem iguais é realizado um pulo de 25 instruções (não é primo)	
11	00001011	MOVCN	10100	1111	00000011	Move o valor 3 para o 16º reg.	
12	00001100	JMPEQ	10101	0101	00010101	Se o valor dos 6º e 16º reg. forem iguais é realizado um pulo de 21 instruções (primo)	
13	00001101	DEC	01000	0101		Decrementa o valor do 6º reg.	
14	00001110	JMP	01111	xxxx	11111010	Retorna 6 instruções (retorna para o CALL de divisão)	
15	00001111	MOVCN	10100	1111	00000000	Move a constante 0 para o 16º registrador	CALL PAR
16	00010000	ADD	00011	0101	0000	Soma os dados do 1º e 16º reg. e guarda no 6º reg.	
17	00010001	SHL	01101	0101	xxxxxxxx		
18	00010010	SHL	01101	0101	xxxxxxxx		
19	00010011	SHL	01101	0101	xxxxxxxx		
20	00010100	SHL	01101	0101	xxxxxxxx		
21	00010101	SHL	01101	0101	xxxxxxxx		
22	00010110	SHL	01101	0101	xxxxxxxx		
23	00010111	SHL	01101	0101	xxxxxxxx		
24	00011000	JMPEQ	10101	0101	00010101	Se o valor dos 6º e 16º reg. forem iguais é realizado um pulo de 21 instruções (não é primo)	
25	00011001	RET	10011				CALL DIVISÃO
26	00011010	MOVCN	10100	1111	00000010	Move o valor 1 para o 16º reg.	
27	00011011	MOVCN	10100	1110	00000000	Move o valor 0 para o 15º reg.	
28	00011100	ADD	00011	0110	0000	Coloca o valor do 1º reg. no 8º reg.	
29	00011101	SUB	00100	0110	0110	Realiza a subtração entre os dados do 1º e do 6º reg. e põe no 7º reg.	
30	00011110	CMP	00101	0111	0110	Informa se o valor do 7º reg. É maior que o do 6º, se sim o valor “00000010” é armazenado no 8º registrador	
31	00011111	JMPEQ	10101	0111	11111110	Se o valor dos 8º e 16º reg. forem iguais é realizado um pulo de 11111110 (subtrai novamente)	É PRIMO
32	00100000	RET	10011				
33	00100001	MOVCN	10100	1010	00110001	Move a constante 39 para o 11º registrador	
34	00100010	MOVA	00010	1010	11101000	Move o valor do 11º reg. Para a posição 232 da memória de dados	
35	00100011	MOVCN	10100	1010	00110000	Move a constante 38 para o 11º registrador	NÃO É PRIMO
36	00100100	MOVA	00010	1010	11101000	Move o valor do 11º reg. Para a posição 232 da memória de dados	

Fonte: Elaborado pelo autor

O algoritmo recebe o número informado pelo usuário na porta E do I/O e executa a sequência de instruções.

3 CONCLUSÃO

Nesta semana finalizamos o projeto da arquitetura do microcontrolador onde cada semana era realizada uma etapa: criação da arquitetura, revisão da arquitetura, criação do projeto RTL e a implementação do projeto RTL. Cumprindo o primeiro projeto requerido na aprendizagem da disciplina de sistemas digitais.

Conseguimos implementar com sucesso o projeto RTL em VHDL seguindo todo o modelo proposto no relatório anterior do grupo 03 com poucas alterações. Com isso conseguimos entender de forma mais aprofundada o funcionamento de um processador e conhecer que existem várias possibilidades de arquitetura. Ao fim de todo o trabalho compreendemos toda a necessidades exigidas na arquitetura de um microcontrolador, passando por todas as reais necessidades exigidas no seu desenvolvimento.

4 REFERÊNCIAS

- [1] VAHID, Frank. **Sistemas digitais: projeto, otimização e HDLs.** Porto Alegre: Artmed, 2008. 560 p.
- [2] STALLINGS, William. **Arquitetura e organização de computadores.** 8. ed. São Paulo: Pearson Prentice Hall, 2010.
- [3] TOCCI, Ronald J.; WIDMER, Neal S.; MOSS, Gregory L.. **Sistemas Digitais: Princípios e Aplicações.** 11^a ed. São Paulo: Pearson, 2011. 830 p.
- [4] Costa, Cesar da. **Elementos de lógica programável com VHDL e DSP: Teoria e Prática.** São Paulo: Érica, 2011. 296 p.

5 ANEXOS

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity Processor is
5     port(CLK_pr, RESET_p: in std_logic;
6             D_p, E_p: in std_logic_vector(7 downto 0);
7             led_ovf_sum_p, led_ovf_mult_p, led_ovf_inc_p: out std_logic;
8             A_p, B_p, C_p: out std_logic_vector(7 downto 0)
9             );
10
11 end Processor;
12
13 architecture Processor_arc of Processor is
14
15 -----
16
17 component registers_bank is
18 port(
19 WDATArb: in std_logic_vector(7 downto 0);
20 WADDRrb, RADDRrb_A, RADDRrb_B: in std_logic_vector(3 downto 0);
21 WRrb, RDrb_A, RDrb_B, CLKrb, CLRrb: in std_logic;
22 RDATArb_A, RDATArb_B: out std_logic_vector(7 downto 0)
23 );
24
25 end component;
26
27 -----
28
29 component ULA is
30     port(A_ula: in std_logic_vector(7 downto 0);
31             B_ula: in std_logic_vector(7 downto 0);
32             Se_ula: in std_logic_vector(3 downto 0);
33             Out_ula: out std_logic_vector(7 downto 0);
34             led_ovf_sum: out std_logic;
35             led_ovf_mult: out std_logic;
36             led_ovf_inc: out std_logic;
37             jump_eq: out std_logic;
38             jump_hi: out std_logic;
39             jump_lo: out std_logic);
40
41 end component;
42
43 -----
44
45 component mux8x6x8 is
46
47 port(
48 IN0, IN1, IN2, IN3, IN4, IN5: in std_logic_vector(7 downto 0);
49 SEL_8x6x8: in std_logic_vector(2 downto 0);
50 OUT_8x6x8: out std_logic_vector(7 downto 0)
51 );
52
53 end component;
54
55 -----
56
57 component Pilha_LIFO is
58     port (PUSH_p, POP_p, CLK_p: in std_logic;
59             Data_in_p : in std_logic_vector(7 downto 0);
60             Data_out_p : OUT std_logic_vector(7 downto 0));
61 end component;
62
63 -----
64
65 component eight_bit_register is
66
```

```
67  port(
68    De: in std_logic_vector(7 downto 0);
69    CLKe, Se, RSe, ENe: in std_logic;
70    Qe: out std_logic_vector(7 downto 0)
71  );
72
73 end component;
74
75 -----
76
77 component contadorPC is
78   port (cont_func: in std_logic_vector(7 downto 0);
79         PC_ld_cont: in std_logic;
80         PC_inc_cont: in std_logic;
81         PC_clr_cont: in std_logic;
82         CLK_PC: in std_logic;
83         pc_out: out std_logic_vector(7 downto 0));
84
85 end component;
86
87 -----
88
89 component IR_register is
90
91   port(
92     De: in std_logic_vector(16 downto 0);
93     CLKe, Se, RSe, ENe: in std_logic;
94     Qe: out std_logic_vector(16 downto 0)
95   );
96
97 end component;
98
99 -----
100
101 component somador8bits is
102   port(A: in std_logic_vector(7 downto 0);
103        B: in std_logic_vector(7 downto 0 );
104        Ci: in std_logic;
105        Co: out std_logic;
106        S_MSB: out std_logic_vector(7 downto 0 )
107      );
108
109 end component;
110
111 -----
112
113 component subtrator8bits is
114   port(A_sub: in std_logic_vector(7 downto 0);
115        B_sub: in std_logic_vector(7 downto 0 );
116        En_sub: in std_logic;
117        S_sub: out std_logic_vector(7 downto 0 )
118      );
119
120 end component;
121
122 -----
123
124 component mux8x3x8 is
125
126   port(
127     IN0, IN1, IN2: in std_logic_vector(7 downto 0);
128     SEL_8x3x8: in std_logic_vector(1 downto 0);
129     OUT_8x3x8: out std_logic_vector(7 downto 0)
130   );
131
132 end component;
```

```
133
134 -----
135 component Control is
136   port(
137     JMP_lo, JMP_hi, JMP_eq, CLKc, RESETc: in std_logic;
138     IR: in std_logic_vector(16 downto 0);
139     IR_ld, I_rd,
140     PC_inc, PC_clr, PC_ld,
141     ld_push_RET, ld_pop_RET,
142     RF_W_wr, RF_Rp_rd, RF_Rq_rd,
143     MEM_rd, MEM_wr,
144     ld_push, ld_pop: out std_logic;
145     selec_PC, F_s: out std_logic_vector(1 downto 0);
146     RF_s: out std_logic_vector(2 downto 0);
147     alu_s, RF_W_addr, RF_Rp_addr, RF_Rq_addr: out std_logic_vector(3 downto 0);
148     JUMP_inc, CALL_addr, Port_Reg_data, MEM_addr: out std_logic_vector(7 downto 0)
149   );
150 end component;
151
152 -----
153
154 COMPONENT I_MEM IS
155   PORT
156   (
157     address      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
158     clock       : IN STD_LOGIC  := '1';
159     q           : OUT STD_LOGIC_VECTOR (16 DOWNTO 0)
160   );
161 END COMPONENT;
162
163 -----
164
165 COMPONENT DATA_MEM IS
166   PORT
167   (
168     address      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
169     clock       : IN STD_LOGIC  := '1';
170     data        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
171     wren        : IN STD_LOGIC ;
172     q           : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
173   );
174 END COMPONENT;
175
176 -----
177
178 signal JMP_lo_SIG, JMP_hi_SIG, JMP_eq_SIG, IR_ld_SIG, I_rd_SIG, PC_inc_SIG, PC_clr_SIG,
179 PC_ld_SIG,
180 ld_push_RET_SIG, ld_pop_RET_SIG, RF_W_wr_SIG, RF_Rp_rd_SIG, RF_Rq_rd_SIG, MEM_rd_SIG,
181 MEM_wr_SIG,
182 ld_push_SIG, ld_pop_SIG, ld_A, ld_B, ld_C: std_logic;
183
184 signal selec_PC_SIG, F_s_SIG: std_logic_vector(1 downto 0);
185
186 signal RF_s_SIG: std_logic_vector(2 downto 0);
187
188 signal alu_s_SIG, RF_W_addr_SIG, RF_Rp_addr_SIG, RF_Rq_addr_SIG: std_logic_vector(3 downto
189 0);
```

```
196
197     signal JUMP_inc_SIG, CALL_addr_SIG, Port_Reg_data_SIG, MEM_addr_SIG, W_data_SIG,
198     Rp_data_SIG, Rq_data_SIG,
199     Out_ula_SIG, LIFO_out_SIG, LIFO_out_RET_SIG, MEM_data_SIG, cont_func_SIG, IMEM_addr_SIG,
200     ADD_JUMP_Out, FULL_JUMP_Out: std_logic_vector(7 downto 0);
201
202 begin
203
204     ld_A <= ((not F_s_SIG(1)) and F_s_SIG(0));
205     ld_B <= (F_s_SIG(1) and (not F_s_SIG(0)));
206     ld_C <= (F_s_SIG(1) and F_s_SIG(0));
207
208     CONTROLE: Control port map(
209
210         --in std_logic;
211
212         JMP_lo => JMP_lo_SIG,
213         JMP_hi => JMP_hi_SIG,
214         JMP_eq => JMP_eq_SIG,
215         CLKc => CLK_pr,
216         RESETc => RESET_p,
217
218         --in std_logic_vector(16 downto 0)
219
220         IR => IR_SIG,
221
222         --out std_logic;
223
224         IR_ld => IR_ld_SIG,
225         I_rd => I_rd_SIG,
226         PC_inc => PC_inc_SIG,
227         PC_clr => PC_clr_SIG,
228         PC_ld => PC_ld_SIG,
229         ld_push_RET => ld_push_RET_SIG,
230         ld_pop_RET => ld_pop_RET_SIG,
231         RF_W_wr => RF_W_wr_SIG,
232         RF_Rp_rd => RF_Rp_rd_SIG,
233         RF_Rq_rd => RF_Rq_rd_SIG,
234         MEM_rd => MEM_rd_SIG,
235         MEM_wr => MEM_wr_SIG,
236         ld_push => ld_push_SIG,
237         ld_pop => ld_pop_SIG,
238
239         --out std_logic_vector(1 downto 0)
240
241         selec_PC => selec_PC_SIG,
242         F_s => F_s_SIG,
243
244         --out std_logic_vector(2 downto 0)
245
246         RF_s => RF_s_SIG,
247
248         --out std_logic_vector(3 downto 0)
249
250         alu_s => alu_s_SIG,
251         RF_W_addr => RF_W_addr_SIG,
252         RF_Rp_addr => RF_Rp_addr_SIG,
253         RF_Rq_addr => RF_Rq_addr_SIG,
254
255         --out std_logic_vector(7 downto 0)
256
257         JUMP_inc => JUMP_inc_SIG,
258         CALL_addr => CALL_addr_SIG,
259         Port_Reg_data => Port_Reg_data_SIG,
```

```
260           MEM_addr => MEM_addr_SIG
261     );
262
263 BANCO: registers_bank port map(
264
265   WDATArb => W_data_SIG,
266   WADDRrb => RF_W_addr_SIG,
267   RADDRrb_A => RF_Rp_addr_SIG,
268   RADDRrb_B => RF_Rq_addr_SIG,
269   WRrb => RF_W_wr_SIG,
270   RDrb_A => RF_Rp_rd_SIG,
271   RDrb_B => RF_Rq_rd_SIG,
272   CLKrb => CLK_pr,
273   CLRrb => '1',
274   RDATArb_A => Rp_data_SIG,
275   RDATArb_B => Rq_data_SIG
276 );
277
278 ALU: ULA port map(
279
280   A_ula => Rp_data_SIG,
281   B_ula => Rq_data_SIG,
282   Se_ula => alu_s_SIG,
283   Out_ula => Out_ula_SIG,
284   led_ovf_sum => led_ovf_sum_p,
285   led_ovf_mult => led_ovf_mult_p,
286   led_ovf_inc => led_ovf_inc_p,
287   jump_eq => JMP_eq_SIG,
288   jump_hi => JMP_hi_SIG,
289   jump_lo => JMP_lo_SIG
290 );
291
292 PILHA: Pilha_LIFO port map(
293
294   PUSH_p => ld_push_SIG,
295   POP_p => ld_pop_SIG,
296   CLK_p => CLK_pr,
297   Data_in_p => Rp_data_SIG,
298   Data_out_p => LIFO_out_SIG
299 );
300
301 MUX_BANCO: mux8x6x8 port map(
302
303   IN0 => D_p,
304   IN1 => E_p,
305   IN2 => LIFO_out_SIG,
306   IN3 => Out_ula_SIG,
307   IN4 => MEM_data_SIG,
308   IN5 => Port_Reg_data_SIG,
309   SEL_8x6x8 => RF_s_SIG,
310   OUT_8x6x8 => W_data_SIG
311 );
312
313
314 REG_A: eight_bit_register port map(
315
316   De => Out_ula_SIG,
317   CLKe => CLK_pr,
318   Se => '1',
319   RSe => '1',
320   ENe => ld_A,
321   QE => A_p
322 );
323
324 REG_B: eight_bit_register port map(
325
```

```
326      De => Out_ula_SIG,  
327      CLKe => CLK_pr,  
328      Se => '1',  
329      RSe => '1',  
330      ENe => ld_B,  
331      Qe => B_p  
332  );  
333  
334  REG_C: eight_bit_register port map(  
335  
336      De => Out_ula_SIG,  
337      CLKe => CLK_pr,  
338      Se => '1',  
339      RSe => '1',  
340      ENe => ld_C,  
341      Qe => C_p  
342  );  
343  
344  IR: IR_register port map(  
345  
346      De => IMEM_data_SIG,  
347      CLKe => CLK_pr,  
348      Se => '1',  
349      RSe => '1',  
350      ENe => IR_ld_SIG,  
351      Qe => IR_SIG  
352  );  
353  
354  PC: contadorPC port map(  
355  
356      cont_func => cont_func_SIG,  
357      PC_ld_cont => PC_ld_SIG,  
358      PC_inc_cont => PC_inc_SIG,  
359      PC_clr_cont => PC_clr_SIG,  
360      CLK_PC => CLK_pr,  
361      pc_out => IMEM_addr_SIG  
362  );  
363  
364  PILHA_JUMP: Pilha_LIFO port map(  
365  
366      PUSH_p => ld_push_RET_SIG,  
367      POP_p => ld_pop_RET_SIG,  
368      CLK_p => CLK_pr,  
369      Data_in_p => IMEM_addr_SIG,  
370      Data_out_p => LIFO_out_RET_SIG  
371  );  
372  
373  ADD_JUMP: somador8bits port map(  
374  
375      A => IMEM_addr_SIG,  
376      B => JUMP_inc_SIG,  
377      Ci => '0',  
378      S_MSB => ADD_JUMP_Out  
379  );  
380  
381  SUB_JUMP: subtrator8bits port map(  
382  
383      A_sub => ADD_JUMP_Out,  
384      B_sub => "00000001",  
385      En_sub => '1',  
386      S_sub => FULL_JUMP_Out  
387  );  
388  
389  MUX_JUMP: mux8x3x8 port map(  
390  
391      IN0 => CALL_addr_SIG,
```

```
392      IN1 => LIFO_out_RET_SIG,  
393      IN2 => FULL_JUMP_Out,  
394      SEL_8x3x8 => selec_PC_SIG,  
395      OUT_8x3x8 => cont_func_SIG  
396  
397  );  
398  
399  INST_MEM: I_MEM port map(  
400      address => IMEM_addr_SIG,  
401      clock => CLK_pr,  
402      q => IMEM_data_SIG  
403  );  
404  
405  DATA_MEMO: DATA_MEM port map(  
406      address => MEM_addr_SIG,  
407      clock => CLK_pr,  
408      data => Rp_data_SIG,  
409      wren => MEM_rd_SIG,  
410      q => MEM_data_SIG  
411  );  
412  
413  );  
414  
415  
416 end Processor_arc;
```

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Control is
5
6  port(
7    JMP_lo, JMP_hi, JMP_eq, CLKc, RESETc: in std_logic;
8
9    IR: in std_logic_vector(16 downto 0);
10
11   IR_ld, I_rd,
12   PC_inc, PC_clr, PC_ld,
13   ld_push_RET, ld_pop_RET,
14   RF_W_wr, RF_Rp_rd, RF_Rq_rd,
15   MEM_rd, MEM_wr,
16   ld_push, ld_pop: out std_logic;
17
18   selec_PC, F_s: out std_logic_vector(1 downto 0);
19
20   RF_s: out std_logic_vector(2 downto 0);
21
22   alu_s, RF_W_addr, RF_Rp_addr, RF_Rq_addr: out std_logic_vector(3 downto 0);
23
24   JUMP_inc, CALL_addr, Port_Reg_data, MEM_addr: out std_logic_vector(7 downto 0)
25 );
26
27 end Control;
28
29 architecture Control_arc of Control is
30
31 type state_c is (START_c, SEARCH_c, DECODE_c, HLT_c, MOVC1_c, MOVC2_c, MOVA_c,
32 ADD_c, SUB_c, CMP_c, MUL_c, INC_c, DEC_c, AND_c, OR_c, XOR_c, NOT_c, SHL_c, SHR_c, JMP_c,
33 PUSH_c, POP_c, CALL_c, RET_c, MOVCN_c, JMPEQ1_c, JMPL01_c, JMPHI1_c, JMPEQ2_c, JMPL02_c,
34 JMPHI2_c,
35 IN_p, OUT_p);
36 signal STATE: state_c;
37 signal IN_sec_sig: std_logic_vector(2 downto 0);
38 begin
39
40 controller_proc: process (CLKc, RESETc)
41 begin
42
43   if RESETc = '1' then
44
45     STATE <= START_c;
46
47   elsif ((CLKc'event) and (CLKc = '1')) then
48
49     case STATE is
50
51       when START_c =>
52         STATE <= SEARCH_c;
53
54       when SEARCH_c =>
55         STATE <= DECODE_c;
56
57       when DECODE_c =>
58         if ((IR(16) = '0') and (IR(15) = '0') and (IR(14) = '0') and (IR(13) ='0')
59         and (IR(12) = '0')) then STATE <= HLT_c;
60         elsif ((IR(16) = '0') and (IR(15) = '0') and (IR(14) = '0') and (IR(13) ='0')
61         and (IR(12) = '1')) then STATE <= MOVC1_c;
62         elsif ((IR(16) = '0') and (IR(15) = '0') and (IR(14) = '0') and (IR(13) ='1')
63         and (IR(12) = '0')) then STATE <= MOVA_c;
64         elsif ((IR(16) = '0') and (IR(15) = '0') and (IR(14) = '0') and (IR(13) ='1')
65         and (IR(12) = '1')) then STATE <= ADD_c;

```

```

62      elsif ((IR(16) = '0') and (IR(15) = '0') and (IR(14) = '1') and (IR(13) ='0')
63      and (IR(12) = '0')) then STATE <= SUB_c;
64      elsif ((IR(16) = '0') and (IR(15) = '0') and (IR(14) = '1') and (IR(13) ='0')
65      and (IR(12) = '1')) then STATE <= CMP_c;
66      elsif ((IR(16) = '0') and (IR(15) = '0') and (IR(14) = '1') and (IR(13) ='1')
67      and (IR(12) = '0')) then STATE <= MUL_c;
68      elsif ((IR(16) = '0') and (IR(15) = '0') and (IR(14) = '1') and (IR(13) ='1')
69      and (IR(12) = '1')) then STATE <= INC_c;
70      elsif ((IR(16) = '0') and (IR(15) = '1') and (IR(14) = '0') and (IR(13) ='0')
71      and (IR(12) = '0')) then STATE <= DEC_c;
72      elsif ((IR(16) = '0') and (IR(15) = '1') and (IR(14) = '0') and (IR(13) ='0')
73      and (IR(12) = '1')) then STATE <= AND_c;
74      elsif ((IR(16) = '0') and (IR(15) = '1') and (IR(14) = '0') and (IR(13) ='1')
75      and (IR(12) = '0')) then STATE <= OR_c;
76      elsif ((IR(16) = '0') and (IR(15) = '1') and (IR(14) = '1') and (IR(13) ='0')
77      and (IR(12) = '1')) then STATE <= XOR_c;
78      elsif ((IR(16) = '0') and (IR(15) = '1') and (IR(14) = '1') and (IR(13) ='1')
79      and (IR(12) = '0')) then STATE <= NOT_c;
80      elsif ((IR(16) = '0') and (IR(15) = '1') and (IR(14) = '1') and (IR(13) ='0')
81      and (IR(12) = '1')) then STATE <= SHL_c;
82      elsif ((IR(16) = '0') and (IR(15) = '1') and (IR(14) = '1') and (IR(13) ='1')
83      and (IR(12) = '0')) then STATE <= SHR_c;
84      elsif ((IR(16) = '0') and (IR(15) = '1') and (IR(14) = '1') and (IR(13) ='1')
85      and (IR(12) = '1')) then STATE <= JMP_c;
86      elsif ((IR(16) = '1') and (IR(15) = '0') and (IR(14) = '0') and (IR(13) ='0')
87      and (IR(12) = '0')) then STATE <= PUSH_c;
88      elsif ((IR(16) = '1') and (IR(15) = '0') and (IR(14) = '0') and (IR(13) ='0')
89      and (IR(12) = '1')) then STATE <= POP_c;
90      elsif ((IR(16) = '1') and (IR(15) = '0') and (IR(14) = '0') and (IR(13) ='1')
91      and (IR(12) = '0')) then STATE <= CALL_c;
92      elsif ((IR(16) = '1') and (IR(15) = '0') and (IR(14) = '0') and (IR(13) ='1')
93      and (IR(12) = '1')) then STATE <= RET_c;
94      elsif ((IR(16) = '1') and (IR(15) = '0') and (IR(14) = '1') and (IR(13) ='0')
95      and (IR(12) = '0')) then STATE <= MOVCN_c;
96      elsif ((IR(16) = '1') and (IR(15) = '0') and (IR(14) = '1') and (IR(13) ='0')
97      and (IR(12) = '1')) then STATE <= JMPEQ1_c;
98      elsif ((IR(16) = '1') and (IR(15) = '0') and (IR(14) = '1') and (IR(13) ='1')
99      and (IR(12) = '0')) then STATE <= JMPLO1_c;
100     elsif ((IR(16) = '1') and (IR(15) = '0') and (IR(14) = '1') and (IR(13) ='1')
101     and (IR(12) = '1')) then STATE <= JMPIH1_c;
102     elsif ((IR(16) = '1') and (IR(15) = '1') and (IR(14) = '0') and (IR(13) ='0')
103     and (IR(12) = '0')) then STATE <= IN_p;
104     elsif ((IR(16) = '1') and (IR(15) = '1') and (IR(14) = '0') and (IR(13) ='0')
105     and (IR(12) = '1')) then STATE <= OUT_p;
106     else
107       STATE <= DECODE_c;
108     end if;
109
110    when HLT_c =>
111      STATE <= HLT_c;
112
113    when MOVC1_c =>
114      STATE <= MOVC2_c;
115
116    when MOVC2_c =>
117      STATE <= SEARCH_c;
118
119    when MOVA_c =>
120      STATE <= SEARCH_c;
121
122    when ADD_c =>
123      STATE <= SEARCH_c;
124
125    when SUB_c =>
126      STATE <= SEARCH_c;

```

```
104      when CMP_c =>
105          STATE <= SEARCH_c;
106
107      when MUL_c =>
108          STATE <= SEARCH_c;
109
110      when INC_c =>
111          STATE <= SEARCH_c;
112
113      when DEC_c =>
114          STATE <= SEARCH_c;
115
116      when AND_c =>
117          STATE <= SEARCH_c;
118
119      when OR_c =>
120          STATE <= SEARCH_c;
121
122      when XOR_c =>
123          STATE <= SEARCH_c;
124
125      when NOT_c =>
126          STATE <= SEARCH_c;
127
128      when SHL_c =>
129          STATE <= SEARCH_c;
130
131      when SHR_c =>
132          STATE <= SEARCH_c;
133
134      when JMP_c =>
135          STATE <= SEARCH_c;
136
137      when PUSH_c =>
138          STATE <= SEARCH_c;
139
140      when POP_c =>
141          STATE <= SEARCH_c;
142
143      when CALL_c =>
144          STATE <= SEARCH_c;
145
146      when RET_c =>
147          STATE <= SEARCH_c;
148
149      when MOVCN_c =>
150          STATE <= SEARCH_c;
151
152
153      when JMPEQ1_c =>
154          if (JMP_eq = '1') then STATE <= JMPEQ2_c;
155          else STATE <= SEARCH_c;
156          end if;
157
158      when JMPL01_c =>
159          if (JMP_lo = '1') then STATE <= JMPL02_c;
160          else STATE <= SEARCH_c;
161          end if;
162
163      when JMPI1_c =>
164          if (JMP_hi = '1') then STATE <= JMPI2_c;
165          else STATE <= SEARCH_c;
166          end if;
167
168      when JMPEQ2_c =>
169          STATE <= SEARCH_c;
```

```

170
171      when JMPLO2_c =>
172          STATE <= SEARCH_c;
173
174      when JMPHI2_c =>
175          STATE <= SEARCH_c;
176
177      when IN_p =>
178          STATE <= SEARCH_c;
179
180      when OUT_p =>
181          STATE <= SEARCH_c;
182
183      end case;
184  end if;
185 end process;
186
187 with STATE select
188
189     IR_ld <= '0' when START_c,
190
191     '1' when SEARCH_c,
192     '0' when DECODE_c,
193     '0' when HLT_c,
194     '0' when MOVC1_c,
195     '0' when MOVC2_c,
196     '0' when MOVA_c,
197     '0' when ADD_c,
198     '0' when SUB_c,
199     '0' when CMP_c,
200     '0' when MUL_c,
201     '0' when INC_c,
202     '0' when DEC_c,
203     '0' when AND_c,
204     '0' when OR_c,
205     '0' when XOR_c,
206     '0' when NOT_c,
207     '0' when SHL_c,
208     '0' when SHR_c,
209     '0' when JMP_c,
210     '0' when PUSH_c,
211     '0' when POP_c,
212     '0' when CALL_c,
213     '0' when RET_c,
214     '0' when MOVCN_c,
215     '0' when JMPEQ1_c,
216     '0' when JMPL01_c,
217     '0' when JMPHI1_c,
218     '0' when JMPEQ2_c,
219     '0' when JMPL02_c,
220     '0' when JMPHI2_c,
221     '0' when IN_p,
222     '0' when OUT_p;
223
224 with STATE select
225
226     I_rd <= '0' when START_c,
227
228     '1' when SEARCH_c,
229     '0' when DECODE_c,
230     '0' when HLT_c,
231     '0' when MOVC1_c,
232     '0' when MOVC2_c,
233     '0' when MOVA_c,
234     '0' when ADD_c,
235     '0' when SUB_c,
236     '0' when CMP_c,
237     '0' when MUL_c,
238     '0' when INC_c,
239     '0' when DEC_c,
```

```

236      '0' when AND_c,
237      '0' when OR_c,
238      '0' when XOR_c,
239      '0' when NOT_c,
240      '0' when SHL_c,
241      '0' when SHR_c,
242      '0' when JMP_c,
243      '0' when PUSH_c,
244      '0' when POP_c,
245      '0' when CALL_c,
246      '0' when RET_c,
247      '0' when MOVCN_c,
248      '0' when JMPEQ1_c,
249      '0' when JMPL01_c,
250      '0' when JMPII1_c,
251      '0' when JMPEQ2_c,
252      '0' when JMPL02_c,
253      '0' when JMPII2_c,
254      '0' when IN_p,
255      '0' when OUT_p;
256
257  with STATE select
258      PC_inc <= '0' when START_c,
259      '1' when SEARCH_c,
260      '0' when DECODE_c,
261      '0' when HLT_c,
262      '0' when MOVC1_c,
263      '0' when MOVC2_c,
264      '0' when MOVA_c,
265      '0' when ADD_c,
266      '0' when SUB_c,
267      '0' when CMP_c,
268      '0' when MUL_c,
269      '0' when INC_c,
270      '0' when DEC_c,
271      '0' when AND_c,
272      '0' when OR_c,
273      '0' when XOR_c,
274      '0' when NOT_c,
275      '0' when SHL_c,
276      '0' when SHR_c,
277      '0' when JMP_c,
278      '0' when PUSH_c,
279      '0' when POP_c,
280      '0' when CALL_c,
281      '0' when RET_c,
282      '0' when MOVCN_c,
283      '0' when JMPEQ1_c,
284      '0' when JMPL01_c,
285      '0' when JMPII1_c,
286      '0' when JMPEQ2_c,
287      '0' when JMPL02_c,
288      '0' when JMPII2_c,
289      '0' when IN_p,
290      '0' when OUT_p;
291
292  with STATE select
293      PC_clr <= '0' when START_c,
294      '1' when SEARCH_c,
295      '1' when DECODE_c,
296      '1' when HLT_c,
297      '1' when MOVC1_c,
298      '1' when MOVC2_c,
299      '1' when MOVA_c,
300      '1' when ADD_c,
301      '1' when SUB_c,

```

```
302      '1' when CMP_C,
303      '1' when MUL_C,
304      '1' when INC_C,
305      '1' when DEC_C,
306      '1' when AND_C,
307      '1' when OR_C,
308      '1' when XOR_C,
309      '1' when NOT_C,
310      '1' when SHL_C,
311      '1' when SHR_C,
312      '1' when JMP_C,
313      '1' when PUSH_C,
314      '1' when POP_C,
315      '1' when CALL_C,
316      '1' when RET_C,
317      '1' when MOVCN_C,
318      '1' when JMPEQ1_C,
319      '1' when JMPL01_C,
320      '1' when JMPII1_C,
321      '1' when JMPEQ2_C,
322      '1' when JMPL02_C,
323      '1' when JMPII2_C,
324      '1' when IN_p,
325      '1' when OUT_p;
326
327  with STATE select
328      PC_ld <= '0' when START_C,
329      '0' when SEARCH_C,
330      '0' when DECODE_C,
331      '0' when HLT_C,
332      '0' when MOVC1_C,
333      '0' when MOVC2_C,
334      '0' when MOVA_C,
335      '0' when ADD_C,
336      '0' when SUB_C,
337      '0' when CMP_C,
338      '0' when MUL_C,
339      '0' when INC_C,
340      '0' when DEC_C,
341      '0' when AND_C,
342      '0' when OR_C,
343      '0' when XOR_C,
344      '0' when NOT_C,
345      '0' when SHL_C,
346      '0' when SHR_C,
347      '1' when JMP_C,
348      '0' when PUSH_C,
349      '0' when POP_C,
350      '1' when CALL_C,
351      '1' when RET_C,
352      '0' when MOVCN_C,
353      '0' when JMPEQ1_C,
354      '0' when JMPL01_C,
355      '0' when JMPII1_C,
356      '1' when JMPEQ2_C,
357      '1' when JMPL02_C,
358      '1' when JMPII2_C,
359      '0' when IN_p,
360      '0' when OUT_p;
361
362  with STATE select
363      ld_push_RET <= '0' when START_C,
364      '0' when SEARCH_C,
365      '0' when DECODE_C,
366      '0' when HLT_C,
367      '0' when MOVC1_C,
```

```

368      '0' when MOVC2_c,
369      '0' when MOVA_c,
370      '0' when ADD_c,
371      '0' when SUB_c,
372      '0' when CMP_c,
373      '0' when MUL_c,
374      '0' when INC_c,
375      '0' when DEC_c,
376      '0' when AND_c,
377      '0' when OR_c,
378      '0' when XOR_c,
379      '0' when NOT_c,
380      '0' when SHL_c,
381      '0' when SHR_c,
382      '0' when JMP_c,
383      '0' when PUSH_c,
384      '0' when POP_c,
385      '1' when CALL_c,
386      '0' when RET_c,
387      '0' when MOVCN_c,
388      '0' when JMPEQ1_c,
389      '0' when JMPL01_c,
390      '0' when JMPII1_c,
391      '0' when JMPEQ2_c,
392      '0' when JMPL02_c,
393      '0' when JMPII2_c,
394      '0' when IN_p,
395      '0' when OUT_p;
396
397  with STATE select
398      ld_pop_RET <= '0' when START_c,
399      '0' when SEARCH_c,
400      '0' when DECODE_c,
401      '0' when HLT_c,
402      '0' when MOVC1_c,
403      '0' when MOVC2_c,
404      '0' when MOVA_c,
405      '0' when ADD_c,
406      '0' when SUB_c,
407      '0' when CMP_c,
408      '0' when MUL_c,
409      '0' when INC_c,
410      '0' when DEC_c,
411      '0' when AND_c,
412      '0' when OR_c,
413      '0' when XOR_c,
414      '0' when NOT_c,
415      '0' when SHL_c,
416      '0' when SHR_c,
417      '0' when JMP_c,
418      '0' when PUSH_c,
419      '0' when POP_c,
420      '0' when CALL_c,
421      '1' when RET_c,
422      '0' when MOVCN_c,
423      '0' when JMPEQ1_c,
424      '0' when JMPL01_c,
425      '0' when JMPII1_c,
426      '0' when JMPEQ2_c,
427      '0' when JMPL02_c,
428      '0' when JMPII2_c,
429      '0' when IN_p,
430      '0' when OUT_p;
431
432  with STATE select
433      RF_W_wr <= '0' when START_c,

```

```

434      '0' when SEARCH_c,
435      '0' when DECODE_c,
436      '0' when HLT_c,
437      '0' when MOVC1_c,
438      '1' when MOVC2_c,
439      '0' when MOVA_c,
440      '1' when ADD_c,
441      '1' when SUB_c,
442      '1' when CMP_c,
443      '1' when MUL_c,
444      '1' when INC_c,
445      '1' when DEC_c,
446      '1' when AND_c,
447      '1' when OR_c,
448      '1' when XOR_c,
449      '1' when NOT_c,
450      '1' when SHL_c,
451      '1' when SHR_c,
452      '0' when JMP_c,
453      '0' when PUSH_c,
454      '1' when POP_c,
455      '0' when CALL_c,
456      '0' when RET_c,
457      '1' when MOVCN_c,
458      '0' when JMPEQ1_c,
459      '0' when JMPL01_c,
460      '0' when JMPII1_c,
461      '0' when JMPEQ2_c,
462      '0' when JMPL02_c,
463      '0' when JMPII2_c,
464      '1' when IN_p,
465      '0' when OUT_p;
466
467 with STATE select
468     RF_Rp_rd <= '0' when START_c,
469     '0' when SEARCH_c,
470     '0' when DECODE_c,
471     '0' when HLT_c,
472     '0' when MOVC1_c,
473     '0' when MOVC2_c,
474     '1' when MOVA_c,
475     '1' when ADD_c,
476     '1' when SUB_c,
477     '1' when CMP_c,
478     '1' when MUL_c,
479     '1' when INC_c,
480     '1' when DEC_c,
481     '1' when AND_c,
482     '1' when OR_c,
483     '1' when XOR_c,
484     '1' when NOT_c,
485     '1' when SHL_c,
486     '1' when SHR_c,
487     '0' when JMP_c,
488     '1' when PUSH_c,
489     '0' when POP_c,
490     '0' when CALL_c,
491     '0' when RET_c,
492     '0' when MOVCN_c,
493     '1' when JMPEQ1_c,
494     '1' when JMPL01_c,
495     '1' when JMPII1_c,
496     '0' when JMPEQ2_c,
497     '0' when JMPL02_c,
498     '0' when JMPII2_c,
499     '0' when IN_p,

```

```
500          '1' when OUT_p;
501
502      with STATE select
503          RF_Rq_rd <= '0' when START_c,
504          '0' when SEARCH_c,
505          '0' when DECODE_c,
506          '0' when HLT_c,
507          '0' when MOVC1_c,
508          '0' when MOVC2_c,
509          '0' when MOVA_c,
510          '1' when ADD_c,
511          '1' when SUB_c,
512          '1' when CMP_c,
513          '1' when MUL_c,
514          '0' when INC_c,
515          '0' when DEC_c,
516          '1' when AND_c,
517          '1' when OR_c,
518          '1' when XOR_c,
519          '0' when NOT_c,
520          '0' when SHL_c,
521          '0' when SHR_c,
522          '0' when JMP_c,
523          '0' when PUSH_c,
524          '0' when POP_c,
525          '0' when CALL_c,
526          '0' when RET_c,
527          '0' when MOVCN_c,
528          '1' when JMPEQ1_c,
529          '1' when JMPL01_c,
530          '1' when JMPHI1_c,
531          '0' when JMPEQ2_c,
532          '0' when JMPL02_c,
533          '0' when JMPHI2_c,
534          '0' when IN_p,
535          '0' when OUT_p;
536
537      with STATE select
538          MEM_rd <= '0' when START_c,
539          '0' when SEARCH_c,
540          '0' when DECODE_c,
541          '0' when HLT_c,
542          '1' when MOVC1_c,
543          '0' when MOVC2_c,
544          '0' when MOVA_c,
545          '0' when ADD_c,
546          '0' when SUB_c,
547          '0' when CMP_c,
548          '0' when MUL_c,
549          '0' when INC_c,
550          '0' when DEC_c,
551          '0' when AND_c,
552          '0' when OR_c,
553          '0' when XOR_c,
554          '0' when NOT_c,
555          '0' when SHL_c,
556          '0' when SHR_c,
557          '0' when JMP_c,
558          '0' when PUSH_c,
559          '0' when POP_c,
560          '0' when CALL_c,
561          '0' when RET_c,
562          '0' when MOVCN_c,
563          '0' when JMPEQ1_c,
564          '0' when JMPL01_c,
565          '0' when JMPHI1_c,
```

```
566      '0' when JMPEQ2_c,
567      '0' when JMPL02_c,
568      '0' when JMPII2_c,
569      '0' when IN_p,
570      '0' when OUT_p;
571
572  with STATE select
573      MEM_wr <= '0' when START_c,
574      '0' when SEARCH_c,
575      '0' when DECODE_c,
576      '0' when HLT_c,
577      '0' when MOVC1_c,
578      '0' when MOVC2_c,
579      '1' when MOVA_c,
580      '0' when ADD_c,
581      '0' when SUB_c,
582      '0' when CMP_c,
583      '0' when MUL_c,
584      '0' when INC_c,
585      '0' when DEC_c,
586      '0' when AND_c,
587      '0' when OR_c,
588      '0' when XOR_c,
589      '0' when NOT_c,
590      '0' when SHL_c,
591      '0' when SHR_c,
592      '0' when JMP_c,
593      '0' when PUSH_c,
594      '0' when POP_c,
595      '0' when CALL_c,
596      '0' when RET_c,
597      '0' when MOVCN_c,
598      '0' when JMPEQ1_c,
599      '0' when JMPL01_c,
600      '0' when JMPII1_c,
601      '0' when JMPEQ2_c,
602      '0' when JMPL02_c,
603      '0' when JMPII2_c,
604      '0' when IN_p,
605      '0' when OUT_p;
606
607  with STATE select
608      ld_push <= '0' when START_c,
609      '0' when SEARCH_c,
610      '0' when DECODE_c,
611      '0' when HLT_c,
612      '0' when MOVC1_c,
613      '0' when MOVC2_c,
614      '0' when MOVA_c,
615      '0' when ADD_c,
616      '0' when SUB_c,
617      '0' when CMP_c,
618      '0' when MUL_c,
619      '0' when INC_c,
620      '0' when DEC_c,
621      '0' when AND_c,
622      '0' when OR_c,
623      '0' when XOR_c,
624      '0' when NOT_c,
625      '0' when SHL_c,
626      '0' when SHR_c,
627      '0' when JMP_c,
628      '1' when PUSH_c,
629      '0' when POP_c,
630      '0' when CALL_c,
631      '0' when RET_c,
```

```
632      '0' when MOVCN_c,
633      '0' when JMPEQ1_c,
634      '0' when JMPL01_c,
635      '0' when JMPII1_c,
636      '0' when JMPEQ2_c,
637      '0' when JMPL02_c,
638      '0' when JMPII2_c,
639      '0' when IN_p,
640      '0' when OUT_p;
641
642  with STATE select
643      ld_pop <= '0' when START_c,
644      '0' when SEARCH_c,
645      '0' when DECODE_c,
646      '0' when HLT_c,
647      '0' when MOVC1_c,
648      '0' when MOVC2_c,
649      '0' when MOVA_c,
650      '0' when ADD_c,
651      '0' when SUB_c,
652      '0' when CMP_c,
653      '0' when MUL_c,
654      '0' when INC_c,
655      '0' when DEC_c,
656      '0' when AND_c,
657      '0' when OR_c,
658      '0' when XOR_c,
659      '0' when NOT_c,
660      '0' when SHL_c,
661      '0' when SHR_c,
662      '0' when JMP_c,
663      '0' when PUSH_c,
664      '1' when POP_c,
665      '0' when CALL_c,
666      '0' when RET_c,
667      '0' when MOVCN_c,
668      '0' when JMPEQ1_c,
669      '0' when JMPL01_c,
670      '0' when JMPII1_c,
671      '0' when JMPEQ2_c,
672      '0' when JMPL02_c,
673      '0' when JMPII2_c,
674      '0' when IN_p,
675      '0' when OUT_p;
676
677  with STATE select
678      selec_PC <= "00" when START_c,
679      "00" when SEARCH_c,
680      "00" when DECODE_c,
681      "00" when HLT_c,
682      "00" when MOVC1_c,
683      "00" when MOVC2_c,
684      "00" when MOVA_c,
685      "00" when ADD_c,
686      "00" when SUB_c,
687      "00" when CMP_c,
688      "00" when MUL_c,
689      "00" when INC_c,
690      "00" when DEC_c,
691      "00" when AND_c,
692      "00" when OR_c,
693      "00" when XOR_c,
694      "00" when NOT_c,
695      "00" when SHL_c,
696      "00" when SHR_c,
697      "10" when JMP_c,
```

```

698      "00" when PUSH_C,
699      "00" when POP_C,
700      "00" when CALL_C,
701      "01" when RET_C,
702      "00" when MOVCN_C,
703      "00" when JMPEQ1_C,
704      "00" when JMPILO1_C,
705      "00" when JMPIH1_C,
706      "10" when JMPEQ2_C,
707      "10" when JMPILO2_C,
708      "10" when JMPIH2_C,
709      "00" when IN_P,
710      "00" when OUT_P;
711
712  with STATE select
713      F_S <= "00" when START_C,
714      "00" when SEARCH_C,
715      "00" when DECODE_C,
716      "00" when HLT_C,
717      "00" when MOVC1_C,
718      "00" when MOVC2_C,
719      "00" when MOVA_C,
720      "00" when ADD_C,
721      "00" when SUB_C,
722      "00" when CMP_C,
723      "00" when MUL_C,
724      "00" when INC_C,
725      "00" when DEC_C,
726      "00" when AND_C,
727      "00" when OR_C,
728      "00" when XOR_C,
729      "00" when NOT_C,
730      "00" when SHL_C,
731      "00" when SHR_C,
732      "00" when JMP_C,
733      "00" when PUSH_C,
734      "00" when POP_C,
735      "00" when CALL_C,
736      "00" when RET_C,
737      "00" when MOVCN_C,
738      "00" when JMPEQ1_C,
739      "00" when JMPILO1_C,
740      "00" when JMPIH1_C,
741      "00" when JMPEQ2_C,
742      "00" when JMPILO2_C,
743      "00" when JMPIH2_C,
744      "00" when IN_P,
745      IR(1 downto 0) when OUT_P;
746
747  with STATE select
748      RF_S <= "000" when START_C,
749      "000" when SEARCH_C,
750      "000" when DECODE_C,
751      "000" when HLT_C,
752      "000" when MOVC1_C,
753      "100" when MOVC2_C,
754      "000" when MOVA_C,
755      "011" when ADD_C,
756      "011" when SUB_C,
757      "011" when CMP_C,
758      "011" when MUL_C,
759      "011" when INC_C,
760      "011" when DEC_C,
761      "011" when AND_C,
762      "011" when OR_C,
763      "011" when XOR_C,

```

```

764      "011" when NOT_c,
765      "011" when SHL_c,
766      "011" when SHR_c,
767      "000" when JMP_c,
768      "000" when PUSH_c,
769      "010" when POP_c,
770      "000" when CALL_c,
771      "000" when RET_c,
772      "101" when MOVCN_c,
773      "000" when JMPEQ1_c,
774      "000" when JMPLO1_c,
775      "000" when JMPII1_c,
776      "000" when JMPEQ2_c,
777      "000" when JMPLO2_c,
778      "000" when JMPII2_c,
779      IN_sec_sig when IN_p,
780      "000" when OUT_p;
781
782  with STATE select
783      alu_s <= "0000" when START_c,
784      "0000" when SEARCH_c,
785      "0000" when DECODE_c,
786      "0000" when HLT_c,
787      "0000" when MOVC1_c,
788      "0000" when MOVC2_c,
789      "0000" when MOVA_c,
790      "0000" when ADD_c,
791      "0001" when SUB_c,
792      "0010" when CMP_c,
793      "0011" when MUL_c,
794      "0100" when INC_c,
795      "0101" when DEC_c,
796      "0110" when AND_c,
797      "0111" when OR_c,
798      "1000" when XOR_c,
799      "1001" when NOT_c,
800      "1010" when SHL_c,
801      "1011" when SHR_c,
802      "0000" when JMP_c,
803      "0000" when PUSH_c,
804      "0000" when POP_c,
805      "0000" when CALL_c,
806      "0000" when RET_c,
807      "0000" when MOVCN_c,
808      "0000" when JMPEQ1_c,
809      "0000" when JMPLO1_c,
810      "0000" when JMPII1_c,
811      "0000" when JMPEQ2_c,
812      "0000" when JMPLO2_c,
813      "0000" when JMPII2_c,
814      "0000" when IN_p,
815      "0000" when OUT_p;
816
817  with STATE select
818      RF_W_addr <= "0000" when START_c,
819      "0000" when SEARCH_c,
820      "0000" when DECODE_c,
821      "0000" when HLT_c,
822      "0000" when MOVC1_c,
823      IR(11 downto 8) when MOVC2_c,
824      "0000" when MOVA_c,
825      IR(11 downto 8) when ADD_c,
826      IR(11 downto 8) when SUB_c,
827      IR(11 downto 8) when CMP_c,
828      IR(11 downto 8) when MUL_c,
829      IR(11 downto 8) when INC_c,

```

```

830      IR(11 downto 8) when DEC_c,
831      IR(11 downto 8) when AND_c,
832      IR(11 downto 8) when OR_c,
833      IR(11 downto 8) when XOR_c,
834      IR(11 downto 8) when NOT_c,
835      IR(11 downto 8) when SHL_c,
836      IR(11 downto 8) when SHR_c,
837      "0000" when JMP_c,
838      "0000" when PUSH_c,
839      IR(11 downto 8) when POP_c,
840      "0000" when CALL_c,
841      "0000" when RET_c,
842      IR(11 downto 8) when MOVCN_c,
843      "0000" when JMPEQ1_c,
844      "0000" when JMPLO1_c,
845      "0000" when JMPIH1_c,
846      "0000" when JMPEQ2_c,
847      "0000" when JMPLO2_c,
848      "0000" when JMPIH2_c,
849      IR(11 downto 8) when IN_p,
850      "0000" when OUT_p;
851
852  with STATE select
853    RF_Rp_addr <= "0000" when START_c,
854    "0000" when SEARCH_c,
855    "0000" when DECODE_c,
856    "0000" when HLT_c,
857    "0000" when MOVC1_c,
858    "0000" when MOVC2_c,
859    IR(11 downto 8) when MOVA_c,
860    IR(7 downto 4) when ADD_c,
861    IR(7 downto 4) when SUB_c,
862    IR(7 downto 4) when CMP_c,
863    IR(7 downto 4) when MUL_c,
864    IR(11 downto 8) when INC_c,
865    IR(11 downto 8) when DEC_c,
866    IR(7 downto 4) when AND_c,
867    IR(7 downto 4) when OR_c,
868    IR(7 downto 4) when XOR_c,
869    IR(7 downto 4) when NOT_c,
870    IR(11 downto 8) when SHL_c,
871    IR(11 downto 8) when SHR_c,
872    "0000" when JMP_c,
873    IR(11 downto 8) when PUSH_c,
874    "0000" when POP_c,
875    "0000" when CALL_c,
876    "0000" when RET_c,
877    "0000" when MOVCN_c,
878    IR(11 downto 8) when JMPEQ1_c,
879    IR(11 downto 8) when JMPLO1_c,
880    IR(11 downto 8) when JMPIH1_c,
881    "0000" when JMPEQ2_c,
882    "0000" when JMPLO2_c,
883    "0000" when JMPIH2_c,
884    IR(11 downto 8) when IN_p,
885    IR(11 downto 8) when OUT_p;
886
887  with STATE select
888    RF_Rq_addr <= "0000" when START_c,
889    "0000" when SEARCH_c,
890    "0000" when DECODE_c,
891    "0000" when HLT_c,
892    "0000" when MOVC1_c,
893    "0000" when MOVC2_c,
894    "0000" when MOVA_c,
895    IR(3 downto 0) when ADD_c,

```

```

896      IR(3 downto 0) when SUB_c,
897      IR(3 downto 0) when CMP_c,
898      IR(3 downto 0) when MUL_c,
899      "0000" when INC_c,
900      "0000" when DEC_c,
901      IR(3 downto 0) when AND_c,
902      IR(3 downto 0) when OR_c,
903      IR(3 downto 0) when XOR_c,
904      "0000" when NOT_c,
905      "0000" when SHL_c,
906      "0000" when SHR_c,
907      "0000" when JMP_c,
908      "0000" when PUSH_c,
909      "0000" when POP_c,
910      "0000" when CALL_c,
911      "0000" when RET_c,
912      "0000" when MOVCN_c,
913      "1111" when JMPEQ1_c,
914      "1111" when JMPL01_c,
915      "1111" when JMPHI1_c,
916      "0000" when JMPEQ2_c,
917      "0000" when JMPL02_c,
918      "0000" when JMPHI2_c,
919      "0000" when IN_p,
920      "0000" when OUT_p;
921
922
923 with STATE select
924     MEM_addr <= "00000000" when START_c,
925     "00000000" when SEARCH_c,
926     "00000000" when DECODE_c,
927     "00000000" when HLT_c,
928     IR(7 downto 0) when MOVC1_c,
929     "00000000" when MOVC2_c,
930     IR(7 downto 0) when MOVA_c,
931     "00000000" when ADD_c,
932     "00000000" when SUB_c,
933     "00000000" when CMP_c,
934     "00000000" when MUL_c,
935     "00000000" when INC_c,
936     "00000000" when DEC_c,
937     "00000000" when AND_c,
938     "00000000" when OR_c,
939     "00000000" when XOR_c,
940     "00000000" when NOT_c,
941     "00000000" when SHL_c,
942     "00000000" when SHR_c,
943     "00000000" when JMP_c,
944     "00000000" when PUSH_c,
945     "00000000" when POP_c,
946     "00000000" when CALL_c,
947     "00000000" when RET_c,
948     "00000000" when MOVCN_c,
949     "00000000" when JMPEQ1_c,
950     "00000000" when JMPL01_c,
951     "00000000" when JMPHI1_c,
952     "00000000" when JMPEQ2_c,
953     "00000000" when JMPL02_c,
954     "00000000" when JMPHI2_c,
955     "00000000" when IN_p,
956     "00000000" when OUT_p;
957
958 with STATE select
959     CALL_addr <= "00000000" when START_c,
960     "00000000" when SEARCH_c,
961     "00000000" when DECODE_c,

```

```

962      "00000000" when HLT_c,
963      "00000000" when MOVC1_c,
964      "00000000" when MOVC2_c,
965      "00000000" when MOVA_c,
966      "00000000" when ADD_c,
967      "00000000" when SUB_c,
968      "00000000" when CMP_c,
969      "00000000" when MUL_c,
970      "00000000" when INC_c,
971      "00000000" when DEC_c,
972      "00000000" when AND_c,
973      "00000000" when OR_c,
974      "00000000" when XOR_c,
975      "00000000" when NOT_c,
976      "00000000" when SHL_c,
977      "00000000" when SHR_c,
978      "00000000" when JMP_c,
979      "00000000" when PUSH_c,
980      "00000000" when POP_c,
981      IR(7 downto 0) when CALL_c,
982      "00000000" when RET_c,
983      "00000000" when MOVCN_c,
984      "00000000" when JMPEQ1_c,
985      "00000000" when JMPL01_c,
986      "00000000" when JMPHI1_c,
987      "00000000" when JMPEQ2_c,
988      "00000000" when JMPL02_c,
989      "00000000" when JMPHI2_c,
990      "00000000" when IN_p,
991      "00000000" when OUT_p;
992
993 with STATE select
994     Port_Reg_data <= "00000000" when START_c,
995     "00000000" when SEARCH_c,
996     "00000000" when DECODE_c,
997     "00000000" when HLT_c,
998     "00000000" when MOVC1_c,
999     "00000000" when MOVC2_c,
1000    "00000000" when MOVA_c,
1001    "00000000" when ADD_c,
1002    "00000000" when SUB_c,
1003    "00000000" when CMP_c,
1004    "00000000" when MUL_c,
1005    "00000000" when INC_c,
1006    "00000000" when DEC_c,
1007    "00000000" when AND_c,
1008    "00000000" when OR_c,
1009    "00000000" when XOR_c,
1010    "00000000" when NOT_c,
1011    "00000000" when SHL_c,
1012    "00000000" when SHR_c,
1013    "00000000" when JMP_c,
1014    "00000000" when PUSH_c,
1015    "00000000" when POP_c,
1016    "00000000" when CALL_c,
1017    "00000000" when RET_c,
1018    IR(7 downto 0) when MOVCN_c,
1019    "00000000" when JMPEQ1_c,
1020    "00000000" when JMPL01_c,
1021    "00000000" when JMPHI1_c,
1022    "00000000" when JMPEQ2_c,
1023    "00000000" when JMPL02_c,
1024    "00000000" when JMPHI2_c,
1025    "00000000" when IN_p,
1026    "00000000" when OUT_p;
1027

```

```
1028  with STATE select
1029      JUMP_inc <= "00000000" when START_c,
1030      "00000000" when SEARCH_c,
1031      "00000000" when DECODE_c,
1032      "00000000" when HLT_c,
1033      "00000000" when MOVC1_c,
1034      "00000000" when MOVC2_c,
1035      "00000000" when MOVA_c,
1036      "00000000" when ADD_c,
1037      "00000000" when SUB_c,
1038      "00000000" when CMP_c,
1039      "00000000" when MUL_c,
1040      "00000000" when INC_c,
1041      "00000000" when DEC_c,
1042      "00000000" when AND_c,
1043      "00000000" when OR_c,
1044      "00000000" when XOR_c,
1045      "00000000" when NOT_c,
1046      "00000000" when SHL_c,
1047      "00000000" when SHR_c,
1048      "00000000" when JMP_c,
1049      "00000000" when PUSH_c,
1050      "00000000" when POP_c,
1051      "00000000" when CALL_c,
1052      "00000000" when RET_c,
1053      "00000000" when MOVCN_c,
1054      "00000000" when JMPEQ1_c,
1055      "00000000" when JMPL01_c,
1056      "00000000" when JMPII1_c,
1057      IR(7 downto 0) when JMPEQ2_c,
1058      IR(7 downto 0) when JMPL02_c,
1059      IR(7 downto 0) when JMPII2_c,
1060      "00000000" when IN_p,
1061      "00000000" when OUT_p;
1062
1063  IN_sec_sig(2) <= '0';
1064  IN_sec_sig(1) <= '0';
1065  IN_sec_sig(0) <= IR(0);
1066
1067 end Control_arc;
```

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity contadorPC is
5      port (cont_func: in std_logic_vector(7 downto 0);
6             PC_ld_cont: in std_logic;
7             PC_inc_cont: in std_logic;
8             PC_clr_cont: in std_logic;
9             CLK_PC: in std_logic;
10            pc_out: out std_logic_vector(7 downto 0));
11
12 end contadorPC;
13
14 architecture arrangement of contadorPC is
15
16     component somador8bits is
17         port(A: in std_logic_vector(7 downto 0);
18               B: in std_logic_vector(7 downto 0 );
19               Ci: in std_logic;
20               Co: out std_logic;
21               S_MSB: out std_logic_vector(7 downto 0));
22
23     end component;
24
25
26     component mux2x1_s is
27         port(A: in std_logic_vector(7 downto 0);
28               B: in std_logic_vector(7 downto 0 );
29               Se: in std_logic ;
30               Sa: out std_logic_vector(7 downto 0));
31
32     end component;
33
34     component eight_bit_register is
35
36         port(
37             De: in std_logic_vector(7 downto 0);
38             CLKe, Se, RSe, ENe: in std_logic;
39             Qe: out std_logic_vector(7 downto 0));
40
41     end component;
42
43     signal signal_out_reg, out_sum, out_mux: std_logic_vector(7 downto 0);
44     signal carry_out: std_logic;
45 begin
46
47
48     somador01: somador8bits port map(
49         A => signal_out_reg,
50         B => "00000001",
51         Ci => '0',
52         Co => carry_out,
53         S_MSB => out_sum );
54
55
56
57     MUX: mux2x1_s port map(
58         A => out_sum,
59         B => cont_func,
60         Se => PC_ld_cont,
61         Sa => out_mux);
62
63     REG: eight_bit_register port map(
64         De => out_mux,
65         CLKe => CLK_PC,
66         Se => '1',

```

```
67      RSe => PC_clr_cont,  
68      ENe => Pc_inc_cont,  
69      QE => signal_out_reg);  
70  
71  
72      pc_out <= signal_out_reg;  
73  
74  
75  end arrangement;
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity registers_bank is
5 port(
6 WDATArb: in std_logic_vector(7 downto 0);
7 WADDRrb, RADDRrb_A, RADDRrb_B: in std_logic_vector(3 downto 0);
8 WRrb, RDrb_A, RDrb_B, CLKrb, CLRrb: in std_logic;
9 RDATArb_A, RDATArb_B: out std_logic_vector(7 downto 0)
10 );
11
12 end registers_bank;
13
14 architecture registers_bank_arc of registers_bank is
15
16 signal OUTd_1x16_aux: std_logic_vector(15 downto 0);
17 signal Qe_aux_0, Qe_aux_1, Qe_aux_2, Qe_aux_3,
18 Qe_aux_4, Qe_aux_5, Qe_aux_6, Qe_aux_7,
19 Qe_aux_8, Qe_aux_9, Qe_aux_10, Qe_aux_11,
20 Qe_aux_12, Qe_aux_13, Qe_aux_14, Qe_aux_15: std_logic_vector(7 downto 0);
21
22 component eight_bit_register is
23
24 port(
25 De: in std_logic_vector(7 downto 0);
26 CLKe, Se, RSe, ENe: in std_logic;
27 Qe: out std_logic_vector(7 downto 0)
28 );
29
30 end component;
31
32 component dec1x16 is
33
34 port(
35 INd_1x16: in std_logic;
36 SELd_1x16: in std_logic_vector(3 downto 0);
37 OUTd_1x16: out std_logic_vector(15 downto 0)
38 );
39
40 end component;
41
42 component mux8x16x8 is
43
44 port(
45 INm_8_0, INm_8_1, INm_8_2, INm_8_3,
46 INm_8_4, INm_8_5, INm_8_6, INm_8_7,
47 INm_8_8, INm_8_9, INm_8_10, INm_8_11,
48 INm_8_12, INm_8_13, INm_8_14, INm_8_15 : in std_logic_vector(7 downto 0);
49 ENm_8x16x8: in std_logic;
50 SELm_8x16x8: in std_logic_vector(3 downto 0);
51 OUTm_8x16x8: out std_logic_vector(7 downto 0)
52 );
53
54 end component;
55
56 begin
57
58 decoder: dec1x16 port map(
59 INd_1x16 => WRrb,
60 SELd_1x16 => WADDRrb,
61 OUTd_1x16 => OUTd_1x16_aux
62 );
63
64 reg0: eight_bit_register port map(
65 De => WDATArb,
66 CLKe => CLKrb,
```

```
67  Se => '1',
68  RSe => CLRrb,
69  ENe => OUTd_1x16_aux(0),
70  Qe => Qe_aux_0
71  );
72
73  reg1: eight_bit_register port map(
74  De => WDATArb,
75  CLKe => CLKrb,
76  Se => '1',
77  RSe => CLRrb,
78  ENe => OUTd_1x16_aux(1),
79  Qe => Qe_aux_1
80  );
81
82  reg2: eight_bit_register port map(
83  De => WDATArb,
84  CLKe => CLKrb,
85  Se => '1',
86  RSe => CLRrb,
87  ENe => OUTd_1x16_aux(2),
88  Qe => Qe_aux_2
89  );
90
91  reg3: eight_bit_register port map(
92  De => WDATArb,
93  CLKe => CLKrb,
94  Se => '1',
95  RSe => CLRrb,
96  ENe => OUTd_1x16_aux(3),
97  Qe => Qe_aux_3
98  );
99
100 reg4: eight_bit_register port map(
101 De => WDATArb,
102 CLKe => CLKrb,
103 Se => '1',
104 RSe => CLRrb,
105 ENe => OUTd_1x16_aux(4),
106 Qe => Qe_aux_4
107 );
108
109 reg5: eight_bit_register port map(
110 De => WDATArb,
111 CLKe => CLKrb,
112 Se => '1',
113 RSe => CLRrb,
114 ENe => OUTd_1x16_aux(5),
115 Qe => Qe_aux_5
116 );
117
118 reg6: eight_bit_register port map(
119 De => WDATArb,
120 CLKe => CLKrb,
121 Se => '1',
122 RSe => CLRrb,
123 ENe => OUTd_1x16_aux(6),
124 Qe => Qe_aux_6
125 );
126
127 reg7: eight_bit_register port map(
128 De => WDATArb,
129 CLKe => CLKrb,
130 Se => '1',
131 RSe => CLRrb,
132 ENe => OUTd_1x16_aux(7),
```

```
133  Qe => Qe_aux_7
134  );
135
136  reg8: eight_bit_register port map(
137  De => WDATArb,
138  CLKe => CLKrb,
139  Se => '1',
140  RSe => CLRrb,
141  ENe => OUTd_1x16_aux(8),
142  Qe => Qe_aux_8
143  );
144
145  reg9: eight_bit_register port map(
146  De => WDATArb,
147  CLKe => CLKrb,
148  Se => '1',
149  RSe => CLRrb,
150  ENe => OUTd_1x16_aux(9),
151  Qe => Qe_aux_9
152  );
153
154  reg10: eight_bit_register port map(
155  De => WDATArb,
156  CLKe => CLKrb,
157  Se => '1',
158  RSe => CLRrb,
159  ENe => OUTd_1x16_aux(10),
160  Qe => Qe_aux_10
161  );
162
163  reg11: eight_bit_register port map(
164  De => WDATArb,
165  CLKe => CLKrb,
166  Se => '1',
167  RSe => CLRrb,
168  ENe => OUTd_1x16_aux(11),
169  Qe => Qe_aux_11
170  );
171
172  reg12: eight_bit_register port map(
173  De => WDATArb,
174  CLKe => CLKrb,
175  Se => '1',
176  RSe => CLRrb,
177  ENe => OUTd_1x16_aux(12),
178  Qe => Qe_aux_12
179  );
180
181  reg13: eight_bit_register port map(
182  De => WDATArb,
183  CLKe => CLKrb,
184  Se => '1',
185  RSe => CLRrb,
186  ENe => OUTd_1x16_aux(13),
187  Qe => Qe_aux_13
188  );
189
190  reg14: eight_bit_register port map(
191  De => WDATArb,
192  CLKe => CLKrb,
193  Se => '1',
194  RSe => CLRrb,
195  ENe => OUTd_1x16_aux(14),
196  Qe => Qe_aux_14
197  );
198
```

```
199  reg15: eight_bit_register port map(
200  De => WDATArb,
201  CLKe => CLKrb,
202  Se => '1',
203  RSe => CLRrb,
204  ENe => OUTd_1x16_aux(15),
205  Qe => Qe_aux_15
206 );
207
208 mux0: mux8x16x8 port map(
209  INm_8_0 => Qe_aux_0,
210  INm_8_1 => Qe_aux_1,
211  INm_8_2 => Qe_aux_2,
212  INm_8_3 => Qe_aux_3,
213  INm_8_4 => Qe_aux_4,
214  INm_8_5 => Qe_aux_5,
215  INm_8_6 => Qe_aux_6,
216  INm_8_7 => Qe_aux_7,
217  INm_8_8 => Qe_aux_8,
218  INm_8_9 => Qe_aux_9,
219  INm_8_10 => Qe_aux_10,
220  INm_8_11 => Qe_aux_11,
221  INm_8_12 => Qe_aux_12,
222  INm_8_13 => Qe_aux_13,
223  INm_8_14 => Qe_aux_14,
224  INm_8_15 => Qe_aux_15,
225  ENm_8x16x8 => RDrb_A,
226  SELm_8x16x8 => RADDRrb_A,
227  OUTm_8x16x8 => RDATArb_A
228 );
229
230
231 mux1: mux8x16x8 port map(
232  INm_8_0 => Qe_aux_0,
233  INm_8_1 => Qe_aux_1,
234  INm_8_2 => Qe_aux_2,
235  INm_8_3 => Qe_aux_3,
236  INm_8_4 => Qe_aux_4,
237  INm_8_5 => Qe_aux_5,
238  INm_8_6 => Qe_aux_6,
239  INm_8_7 => Qe_aux_7,
240  INm_8_8 => Qe_aux_8,
241  INm_8_9 => Qe_aux_9,
242  INm_8_10 => Qe_aux_10,
243  INm_8_11 => Qe_aux_11,
244  INm_8_12 => Qe_aux_12,
245  INm_8_13 => Qe_aux_13,
246  INm_8_14 => Qe_aux_14,
247  INm_8_15 => Qe_aux_15,
248  ENm_8x16x8 => RDrb_B,
249  SELm_8x16x8 => RADDRrb_B,
250  OUTm_8x16x8 => RDATArb_B
251 );
252
253 end registers_bank_arc;
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Cont_updW_4B is
5      port (PRE, CLR: in std_logic_vector(3 downto 0);
6             UD, CK, EN: IN std_logic ;
7             S : OUT std_logic_vector(3 downto 0));
8  end Cont_updW_4B;
9
10 architecture circuito of Cont_updW_4B is
11     component ffjk IS
12         port (clk ,J ,K ,P , C : IN std_logic ;
13                q : OUT std_logic );
14     end component;
15
16     signal q_B01, q_B02, q_B03, or_B01, or_B02, or_B03: std_logic;
17     signal and_B01, and_B02, and_B03: std_logic_vector(1 downto 0);
18     signal pres: std_logic_vector(3 downto 0);
19
20 begin
21
22     B01: ffjk port map(
23         clk => CK,
24         P => PRE(0),
25         C => CLR(0),
26         J => EN,
27         K => EN,
28         q => q_B01);
29
30     and_B01(0) <= q_B01 AND UD AND EN;
31     and_B01(1) <= (not q_B01) AND (not UD) AND EN;
32     or_B01 <= and_B01(0) OR and_B01(1);
33
34     B02: ffjk port map(
35         clk => CK,
36         P => PRE(1),
37         C => CLR(1),
38         J => or_B01,
39         K => or_B01,
40         q => q_B02);
41
42     and_B02(0) <= q_B02 AND and_B01(0);
43     and_B02(1) <= (not q_B02) AND and_B01(1);
44     or_B02 <= and_B02(0) OR and_B02(1);
45
46     B03: ffjk port map(
47         clk => CK,
48         P => PRE(2),
49         C => CLR(2),
50         J => or_B02,
51         K => or_B02,
52         q => q_B03);
53
54     and_B03(0) <= q_B03 AND and_B02(0);
55     and_B03(1) <= (not q_B03) AND and_B02(1);
56     or_B03 <= and_B03(0) OR and_B03(1);
57
58     B04: ffjk port map(
59         clk => CK,
60         P => PRE(3),
61         C => CLR(3),
62         J => or_B03,
63         K => or_B03,
64         q => S(3));
65
66     S(0) <= q_B01;
```

```
67      S (1)  <= q_B02;  
68      S (2)  <= q_B03;  
69  
70  end circuito;
```

```
1  -- megafunction wizard: %RAM: 1-PORT%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: altsyncram
5
6  -- =====
7  -- File Name: DATA_MEM.vhd
8  -- Megafunction Name(s):
9  --      altsyncram
10 --
11 -- Simulation Library Files(s):
12 --      altera_mf
13 -- =====
14 -- ****
15 -- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
16 --
17 -- 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
18 -- ****
19
20
21 --Copyright (C) 1991-2013 Altera Corporation
22 --Your use of Altera Corporation's design tools, logic functions
23 --and other software and tools, and its AMPP partner logic
24 --functions, and any output files from any of the foregoing
25 --(including device programming or simulation files), and any
26 --associated documentation or information are expressly subject
27 --to the terms and conditions of the Altera Program License
28 --Subscription Agreement, Altera MegaCore Function License
29 --Agreement, or other applicable license agreement, including,
30 --without limitation, that your use is for the sole purpose of
31 --programming logic devices manufactured by Altera and sold by
32 --Altera or its authorized distributors. Please refer to the
33 --applicable agreement for further details.
34
35
36 LIBRARY ieee;
37 USE ieee.std_logic_1164.all;
38
39 LIBRARY altera_mf;
40 USE altera_mf.all;
41
42 ENTITY DATA_MEM IS
43     PORT
44     (
45         address      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
46         clock        : IN STD_LOGIC := '1';
47         data         : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
48         wren         : IN STD_LOGIC ;
49         q            : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
50     );
51 END DATA_MEM;
52
53
54 ARCHITECTURE SYN OF data_mem IS
55
56     SIGNAL sub_wire0  : STD_LOGIC_VECTOR (7 DOWNTO 0);
57
58
59     COMPONENT altsyncram
60     GENERIC (
61         clock_enable_input_a      : STRING;
62         clock_enable_output_a     : STRING;
63         init_file                : STRING;
64         intended_device_family    : STRING;
65         lpm_hint                 : STRING;
```

```

67      lpm_type      : STRING;
68      numwords_a    : NATURAL;
69      operation_mode : STRING;
70      outdata_aclr_a : STRING;
71      outdata_reg_a  : STRING;
72      power_up_uninitialized : STRING;
73      widthad_a     : NATURAL;
74      width_a        : NATURAL;
75      width_byteena_a : NATURAL
76  );
77  PORT (
78      address_a   : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
79      clock0      : IN STD_LOGIC ;
80      data_a       : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
81      wren_a       : IN STD_LOGIC ;
82      q_a          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
83  );
84 END COMPONENT;
85
86 BEGIN
87     q    <= sub_wire0(7 DOWNTO 0);
88
89     altsyncram_component : altsyncram
90     GENERIC MAP (
91         clock_enable_input_a => "BYPASS",
92         clock_enable_output_a => "BYPASS",
93         init_file => "DATA_MEM.mif",
94         intended_device_family => "Cyclone II",
95         lpm_hint => "ENABLE_RUNTIME_MOD=NO",
96         lpm_type => "altsyncram",
97         numwords_a => 256,
98         operation_mode => "SINGLE_PORT",
99         outdata_aclr_a => "NONE",
100        outdata_reg_a => "UNREGISTERED",
101        power_up_uninitialized => "FALSE",
102        widthad_a => 8,
103        width_a => 8,
104        width_byteena_a => 1
105    )
106    PORT MAP (
107        address_a => address,
108        clock0 => clock,
109        data_a => data,
110        wren_a => wren,
111        q_a => sub_wire0
112    );
113
114
115
116 END SYN;
117
118 -- =====
119 -- CNX file retrieval info
120 -- =====
121 -- Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
122 -- Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
123 -- Retrieval info: PRIVATE: AclrByte NUMERIC "0"
124 -- Retrieval info: PRIVATE: AclrData NUMERIC "0"
125 -- Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
126 -- Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
127 -- Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
128 -- Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
129 -- Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
130 -- Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
131 -- Retrieval info: PRIVATE: Clken NUMERIC "0"
132 -- Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"

```

```
133 -- Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
134 -- Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
135 -- Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
136 -- Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
137 -- Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
138 -- Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
139 -- Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
140 -- Retrieval info: PRIVATE: MIFFilename STRING "DATA_MEM.mif"
141 -- Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "256"
142 -- Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
143 -- Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
144 -- Retrieval info: PRIVATE: RegAddr NUMERIC "1"
145 -- Retrieval info: PRIVATE: RegData NUMERIC "1"
146 -- Retrieval info: PRIVATE: RegOutput NUMERIC "0"
147 -- Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
148 -- Retrieval info: PRIVATE: SingleClock NUMERIC "1"
149 -- Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
150 -- Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
151 -- Retrieval info: PRIVATE: WidthAddr NUMERIC "8"
152 -- Retrieval info: PRIVATE: WidthData NUMERIC "8"
153 -- Retrieval info: PRIVATE: rden NUMERIC "0"
154 -- Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
155 -- Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
156 -- Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
157 -- Retrieval info: CONSTANT: INIT_FILE STRING "DATA_MEM.mif"
158 -- Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
159 -- Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
160 -- Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
161 -- Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "256"
162 -- Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
163 -- Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
164 -- Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
165 -- Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
166 -- Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "8"
167 -- Retrieval info: CONSTANT: WIDTH_A NUMERIC "8"
168 -- Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
169 -- Retrieval info: USED_PORT: address 0 0 8 0 INPUT NODEFVAL "address[7..0]"
170 -- Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
171 -- Retrieval info: USED_PORT: data 0 0 8 0 INPUT NODEFVAL "data[7..0]"
172 -- Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL "q[7..0]"
173 -- Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
174 -- Retrieval info: CONNECT: @address_a 0 0 8 0 address 0 0 8 0
175 -- Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
176 -- Retrieval info: CONNECT: @data_a 0 0 8 0 data 0 0 8 0
177 -- Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
178 -- Retrieval info: CONNECT: q 0 0 8 0 @q_a 0 0 8 0
179 -- Retrieval info: GEN_FILE: TYPE_NORMAL DATA_MEM.vhd TRUE
180 -- Retrieval info: GEN_FILE: TYPE_NORMAL DATA_MEM.inc FALSE
181 -- Retrieval info: GEN_FILE: TYPE_NORMAL DATA_MEM.cmp FALSE
182 -- Retrieval info: GEN_FILE: TYPE_NORMAL DATA_MEM.bsf FALSE
183 -- Retrieval info: GEN_FILE: TYPE_NORMAL DATA_MEM_inst.vhd FALSE
184 -- Retrieval info: LIB_FILE: altera_mf
185
```

```
1  -- megafunction wizard: %ROM: 1-PORT%
2  -- GENERATION: STANDARD
3  -- VERSION: WM1.0
4  -- MODULE: altsyncram
5
6  -- =====
7  -- File Name: I_MEM.vhd
8  -- Megafunction Name(s):
9  --      altsyncram
10 --
11 -- Simulation Library Files(s):
12 --      altera_mf
13 -- =====
14 -- ****
15 -- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
16 --
17 -- 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
18 -- ****
19
20
21 --Copyright (C) 1991-2013 Altera Corporation
22 --Your use of Altera Corporation's design tools, logic functions
23 --and other software and tools, and its AMPP partner logic
24 --functions, and any output files from any of the foregoing
25 --(including device programming or simulation files), and any
26 --associated documentation or information are expressly subject
27 --to the terms and conditions of the Altera Program License
28 --Subscription Agreement, Altera MegaCore Function License
29 --Agreement, or other applicable license agreement, including,
30 --without limitation, that your use is for the sole purpose of
31 --programming logic devices manufactured by Altera and sold by
32 --Altera or its authorized distributors. Please refer to the
33 --applicable agreement for further details.
34
35
36 LIBRARY ieee;
37 USE ieee.std_logic_1164.all;
38
39 LIBRARY altera_mf;
40 USE altera_mf.all;
41
42 ENTITY I_MEM IS
43   PORT
44   (
45     address      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
46     clock       : IN STD_LOGIC  := '1';
47     q           : OUT STD_LOGIC_VECTOR (16 DOWNTO 0)
48   );
49 END I_MEM;
50
51
52 ARCHITECTURE SYN OF i_mem IS
53
54   SIGNAL sub_wire0  : STD_LOGIC_VECTOR (16 DOWNTO 0);
55
56
57
58   COMPONENT altsyncram
59   GENERIC (
60     clock_enable_input_a    : STRING;
61     clock_enable_output_a   : STRING;
62     init_file               : STRING;
63     intended_device_family  : STRING;
64     lpm_hint                : STRING;
65     lpm_type                : STRING;
66     numwords_a              : NATURAL;
```

```

67      operation_mode      : STRING;
68      outdata_aclr_a      : STRING;
69      outdata_reg_a       : STRING;
70      widthad_a          : NATURAL;
71      width_a             : NATURAL;
72      width_byteena_a    : NATURAL
73  );
74  PORT (
75      address_a      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
76      clock0         : IN STD_LOGIC ;
77      q_a            : OUT STD_LOGIC_VECTOR (16 DOWNTO 0)
78  );
79 END COMPONENT;
80
81 BEGIN
82     q    <= sub_wire0(16 DOWNTO 0);
83
84     altsyncram_component : altsyncram
85     GENERIC MAP (
86         clock_enable_input_a => "BYPASS",
87         clock_enable_output_a => "BYPASS",
88         init_file => "I_MEM.mif",
89         intended_device_family => "Cyclone II",
90         lpm_hint => "ENABLE_RUNTIME_MOD=NO",
91         lpm_type => "altsyncram",
92         numwords_a => 256,
93         operation_mode => "ROM",
94         outdata_aclr_a => "NONE",
95         outdata_reg_a => "UNREGISTERED",
96         widthad_a => 8,
97         width_a => 17,
98         width_byteena_a => 1
99     )
100    PORT MAP (
101        address_a => address,
102        clock0 => clock,
103        q_a => sub_wire0
104    );
105
106
107
108 END SYN;
109
110 -- =====
111 -- CNX file retrieval info
112 -- =====
113 -- Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
114 -- Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
115 -- Retrieval info: PRIVATE: AclrByte NUMERIC "0"
116 -- Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
117 -- Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
118 -- Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
119 -- Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
120 -- Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
121 -- Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
122 -- Retrieval info: PRIVATE: Clken NUMERIC "0"
123 -- Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
124 -- Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
125 -- Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
126 -- Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
127 -- Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
128 -- Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
129 -- Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
130 -- Retrieval info: PRIVATE: MIFfilename STRING "I_MEM.mif"
131 -- Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "256"
132 -- Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"

```

```
133 -- Retrieval info: PRIVATE: RegAddr NUMERIC "1"
134 -- Retrieval info: PRIVATE: RegOutput NUMERIC "0"
135 -- Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
136 -- Retrieval info: PRIVATE: SingleClock NUMERIC "1"
137 -- Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
138 -- Retrieval info: PRIVATE: WidthAddr NUMERIC "8"
139 -- Retrieval info: PRIVATE: WidthData NUMERIC "17"
140 -- Retrieval info: PRIVATE: rden NUMERIC "0"
141 -- Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
142 -- Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
143 -- Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
144 -- Retrieval info: CONSTANT: INIT_FILE STRING "I_MEM.mif"
145 -- Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
146 -- Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
147 -- Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
148 -- Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "256"
149 -- Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
150 -- Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
151 -- Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
152 -- Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "8"
153 -- Retrieval info: CONSTANT: WIDTH_A NUMERIC "17"
154 -- Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
155 -- Retrieval info: USED_PORT: address 0 0 8 0 INPUT NODEFVAL "address[7..0]"
156 -- Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
157 -- Retrieval info: USED_PORT: q 0 0 17 0 OUTPUT NODEFVAL "q[16..0]"
158 -- Retrieval info: CONNECT: @address_a 0 0 8 0 address 0 0 8 0
159 -- Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
160 -- Retrieval info: CONNECT: q 0 0 17 0 @q_a 0 0 17 0
161 -- Retrieval info: GEN_FILE: TYPE_NORMAL I_MEM.vhd TRUE
162 -- Retrieval info: GEN_FILE: TYPE_NORMAL I_MEM.inc FALSE
163 -- Retrieval info: GEN_FILE: TYPE_NORMAL I_MEM.cmp FALSE
164 -- Retrieval info: GEN_FILE: TYPE_NORMAL I_MEM.bsf FALSE
165 -- Retrieval info: GEN_FILE: TYPE_NORMAL I_MEM_inst.vhd FALSE
166 -- Retrieval info: LIB_FILE: altera_mf
167
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity dec1x16 is
5
6 port(
7 INd_1x16: in std_logic;
8 SELd_1x16: in std_logic_vector(3 downto 0);
9 OUTd_1x16: out std_logic_vector(15 downto 0)
10 );
11
12 end dec1x16;
13
14 architecture dec1x16_arc of dec1x16 is
15
16 begin
17
18 OUTd_1x16(0) <= (INd_1x16 and (not SELd_1x16(3)) and (not SELd_1x16(2)) and (not SELd_1x16(1)) and (not SELd_1x16(0)));
19 OUTd_1x16(1) <= (INd_1x16 and (not SELd_1x16(3)) and (not SELd_1x16(2)) and (not SELd_1x16(1)) and (SELd_1x16(0)));
20 OUTd_1x16(2) <= (INd_1x16 and (not SELd_1x16(3)) and (not SELd_1x16(2)) and (SELd_1x16(1)) and (not SELd_1x16(0)));
21 OUTd_1x16(3) <= (INd_1x16 and (not SELd_1x16(3)) and (not SELd_1x16(2)) and (SELd_1x16(1)) and (SELd_1x16(0)));
22 OUTd_1x16(4) <= (INd_1x16 and (not SELd_1x16(3)) and (SELd_1x16(2)) and (not SELd_1x16(1)) and (not SELd_1x16(0)));
23 OUTd_1x16(5) <= (INd_1x16 and (not SELd_1x16(3)) and (SELd_1x16(2)) and (not SELd_1x16(1)) and (SELd_1x16(0)));
24 OUTd_1x16(6) <= (INd_1x16 and (not SELd_1x16(3)) and (SELd_1x16(2)) and (SELd_1x16(1)) and (not SELd_1x16(0)));
25 OUTd_1x16(7) <= (INd_1x16 and (not SELd_1x16(3)) and (SELd_1x16(2)) and (SELd_1x16(1)) and (SELd_1x16(0)));
26 OUTd_1x16(8) <= (INd_1x16 and (SELd_1x16(3)) and (not SELd_1x16(2)) and (not SELd_1x16(1)) and (not SELd_1x16(0)));
27 OUTd_1x16(9) <= (INd_1x16 and (SELd_1x16(3)) and (not SELd_1x16(2)) and (not SELd_1x16(1)) and (SELd_1x16(0)));
28 OUTd_1x16(10) <= (INd_1x16 and (SELd_1x16(3)) and (not SELd_1x16(2)) and (SELd_1x16(1)) and (not SELd_1x16(0)));
29 OUTd_1x16(11) <= (INd_1x16 and (SELd_1x16(3)) and (not SELd_1x16(2)) and (SELd_1x16(1)) and (SELd_1x16(0)));
30 OUTd_1x16(12) <= (INd_1x16 and (SELd_1x16(3)) and (SELd_1x16(2)) and (not SELd_1x16(1)) and (not SELd_1x16(0)));
31 OUTd_1x16(13) <= (INd_1x16 and (SELd_1x16(3)) and (SELd_1x16(2)) and (not SELd_1x16(1)) and (SELd_1x16(0)));
32 OUTd_1x16(14) <= (INd_1x16 and (SELd_1x16(3)) and (SELd_1x16(2)) and (SELd_1x16(1)) and (not SELd_1x16(0)));
33 OUTd_1x16(15) <= (INd_1x16 and (SELd_1x16(3)) and (SELd_1x16(2)) and (SELd_1x16(1)) and (SELd_1x16(0)));
34
35 end dec1x16_arc;
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity FFD is
5 port(
6
7 D, CLK, S, RS: in std_logic;
8 Q: out std_logic
9 );
10
11 end FFD;
12
13 architecture FFD_arc of FFD is
14
15 signal qr: std_logic;
16
17 begin
18
19 process(CLK, S, RS)
20 begin
21
22 if RS = '0' then qr <= '0';
23 elsif S = '0' then qr <= '1';
24 elsif (CLK'EVENT) and (CLK = '1') then
25     qr <= d;
26 end if;
27 end process;
28
29 Q <= qr;
30
31 end FFD_arc;
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 ENTITY ffjk IS
5     port ( clk ,J ,K ,P , C : IN std_logic ;
6             q : OUT std_logic );
7 END ffjk ;
8
9 ARCHITECTURE ckt OF ffjk IS
10 SIGNAL qS : std_logic ;
11 BEGIN
12     PROCESS ( clk ,P , C )
13     BEGIN
14         IF P = '0' THEN qS <= '1';
15         ELSIF C = '0' THEN qS <= '0';
16         ELSIF clk = '1' AND clk ' EVENT THEN
17             IF J = '1' AND K = '1' THEN qS <= NOT qS ;
18             ELSIF J = '1' AND K = '0' THEN qS <= '1';
19             ELSIF J = '0' AND K = '1' THEN qS <= '0';
20             END IF ;
21         END IF ;
22     END PROCESS ;
23     q <= qS ;
24 END ckt ;
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity IR_register is
5
6  port(
7    De: in std_logic_vector(16 downto 0);
8    CLKe, Se, RSe, ENe: in std_logic;
9    Qe: out std_logic_vector(16 downto 0)
10 );
11
12 end IR_register;
13
14 architecture IR_register_arc of IR_register is
15
16 component FFD is
17 port(
18
19   D, CLK, S, RS: in std_logic;
20   Q: out std_logic
21 );
22
23 end component;
24
25 component mux34x17 is
26
27 port(
28   Mm1, Mm2: in std_logic_vector(16 downto 0);
29   Sm: in std_logic;
30   Xm: out std_logic_vector(16 downto 0)
31 );
32
33 end component;
34
35 signal Dem, Qer: std_logic_vector(16 downto 0);
36
37 begin
38
39 mult: mux34x17 port map(
40   Mm1 => Qer,
41   Mm2 => De,
42   Sm => ENe,
43   Xm => Dem
44 );
45
46 reg0: FFD port map(
47   D => Dem(0),
48   CLK => CLKe,
49   S => Se,
50   RS => RSe,
51   Q => Qer(0)
52 );
53
54 reg1: FFD port map(
55   D => Dem(1),
56   CLK => CLKe,
57   S => Se,
58   RS => RSe,
59   Q => Qer(1)
60 );
61
62 reg2: FFD port map(
63   D => Dem(2),
64   CLK => CLKe,
65   S => Se,
66   RS => RSe,
```

```
67      Q => Qer(2)
68  );
69
70  reg3: FFD port map(
71      D => Dem(3),
72      CLK => CLKe,
73      S => Se,
74      RS => RSe,
75      Q => Qer(3)
76  );
77
78  reg4: FFD port map(
79      D => Dem(4),
80      CLK => CLKe,
81      S => Se,
82      RS => RSe,
83      Q => Qer(4)
84  );
85
86  reg5: FFD port map(
87      D => Dem(5),
88      CLK => CLKe,
89      S => Se,
90      RS => RSe,
91      Q => Qer(5)
92  );
93
94  reg6: FFD port map(
95      D => Dem(6),
96      CLK => CLKe,
97      S => Se,
98      RS => RSe,
99      Q => Qer(6)
100 );
101
102 reg7: FFD port map(
103     D => Dem(7),
104     CLK => CLKe,
105     S => Se,
106     RS => RSe,
107     Q => Qer(7)
108 );
109
110 reg8: FFD port map(
111     D => Dem(8),
112     CLK => CLKe,
113     S => Se,
114     RS => RSe,
115     Q => Qer(8)
116 );
117
118 reg9: FFD port map(
119     D => Dem(9),
120     CLK => CLKe,
121     S => Se,
122     RS => RSe,
123     Q => Qer(9)
124 );
125
126 reg10: FFD port map(
127     D => Dem(10),
128     CLK => CLKe,
129     S => Se,
130     RS => RSe,
131     Q => Qer(10)
132 );
```

```
133
134     reg11: FFD port map(
135         D => Dem(11),
136         CLK => CLKE,
137         S => Se,
138         RS => RSe,
139         Q => Qer(11)
140     );
141
142     reg12: FFD port map(
143         D => Dem(12),
144         CLK => CLKE,
145         S => Se,
146         RS => RSe,
147         Q => Qer(12)
148     );
149
150     reg13: FFD port map(
151         D => Dem(13),
152         CLK => CLKE,
153         S => Se,
154         RS => RSe,
155         Q => Qer(13)
156     );
157
158     reg14: FFD port map(
159         D => Dem(14),
160         CLK => CLKE,
161         S => Se,
162         RS => RSe,
163         Q => Qer(14)
164     );
165
166     reg15: FFD port map(
167         D => Dem(15),
168         CLK => CLKE,
169         S => Se,
170         RS => RSe,
171         Q => Qer(15)
172     );
173
174     reg16: FFD port map(
175         D => Dem(16),
176         CLK => CLKE,
177         S => Se,
178         RS => RSe,
179         Q => Qer(16)
180     );
181
182     Qe <= Qer;
183
184 end IR_register_arc;
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity mux2x1 is
5 port(
6 M1, M2, S: in std_logic;
7 X: out std_logic
8 );
9 end mux2x1;
10
11 architecture mux2x1_arc of mux2x1 is
12 begin
13 X <= ((not S) and M1) or (S and M2);
14 end mux2x1_arc;
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity mux2x1_s is
5      port(A: in std_logic_vector(7 downto 0);
6            B: in std_logic_vector(7 downto 0);
7            Se: in std_logic ;
8            Sa: out std_logic_vector(7 downto 0));
9
10 end mux2x1_s;
11
12 architecture funcionamento of mux2x1_s is
13 begin
14     Sa(7)  <= (A(7) and (not Se)) or (B(7) and Se);
15     Sa(6)  <= (A(6) and (not Se)) or (B(6) and Se);
16     Sa(5)  <= (A(5) and (not Se)) or (B(5) and Se);
17     Sa(4)  <= (A(4) and (not Se)) or (B(4) and Se);
18     Sa(3)  <= (A(3) and (not Se)) or (B(3) and Se);
19     Sa(2)  <= (A(2) and (not Se)) or (B(2) and Se);
20     Sa(1)  <= (A(1) and (not Se)) or (B(1) and Se);
21     Sa(0)  <= (A(0) and (not Se)) or (B(0) and Se);
22
23 end funcionamento;
24
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity mux8x3x8 is
5
6 port(
7 IN0, IN1, IN2: in std_logic_vector(7 downto 0);
8 SEL_8x3x8: in std_logic_vector(1 downto 0);
9 OUT_8x3x8: out std_logic_vector(7 downto 0)
10 );
11
12 end mux8x3x8;
13
14 architecture mux8x3x8_arc of mux8x3x8 is
15
16 signal nSEL_8x3x8: std_logic_vector(1 downto 0);
17
18 begin
19
20 nSEL_8x3x8(1) <= not SEL_8x3x8(1);
21 nSEL_8x3x8(0) <= not SEL_8x3x8(0);
22
23 OUT_8x3x8(7) <=
24 ((IN0(7) and nSEL_8x3x8(1) and nSEL_8x3x8(0)) or
25 (IN1(7) and nSEL_8x3x8(1) and SEL_8x3x8(0)) or
26 (IN2(7) and SEL_8x3x8(1) and nSEL_8x3x8(0)));
27
28 OUT_8x3x8(6) <=
29 ((IN0(6) and nSEL_8x3x8(1) and nSEL_8x3x8(0)) or
30 (IN1(6) and nSEL_8x3x8(1) and SEL_8x3x8(0)) or
31 (IN2(6) and SEL_8x3x8(1) and nSEL_8x3x8(0)));
32
33 OUT_8x3x8(5) <=
34 ((IN0(5) and nSEL_8x3x8(1) and nSEL_8x3x8(0)) or
35 (IN1(5) and nSEL_8x3x8(1) and SEL_8x3x8(0)) or
36 (IN2(5) and SEL_8x3x8(1) and nSEL_8x3x8(0)));
37
38 OUT_8x3x8(4) <=
39 ((IN0(4) and nSEL_8x3x8(1) and nSEL_8x3x8(0)) or
40 (IN1(4) and nSEL_8x3x8(1) and SEL_8x3x8(0)) or
41 (IN2(4) and SEL_8x3x8(1) and nSEL_8x3x8(0)));
42
43 OUT_8x3x8(3) <=
44 ((IN0(3) and nSEL_8x3x8(1) and nSEL_8x3x8(0)) or
45 (IN1(3) and nSEL_8x3x8(1) and SEL_8x3x8(0)) or
46 (IN2(3) and SEL_8x3x8(1) and nSEL_8x3x8(0)));
47
48 OUT_8x3x8(2) <=
49 ((IN0(2) and nSEL_8x3x8(1) and nSEL_8x3x8(0)) or
50 (IN1(2) and nSEL_8x3x8(1) and SEL_8x3x8(0)) or
51 (IN2(2) and SEL_8x3x8(1) and nSEL_8x3x8(0)));
52
53 OUT_8x3x8(1) <=
54 ((IN0(1) and nSEL_8x3x8(1) and nSEL_8x3x8(0)) or
55 (IN1(1) and nSEL_8x3x8(1) and SEL_8x3x8(0)) or
56 (IN2(1) and SEL_8x3x8(1) and nSEL_8x3x8(0)));
57
58 OUT_8x3x8(0) <=
59 ((IN0(0) and nSEL_8x3x8(1) and nSEL_8x3x8(0)) or
60 (IN1(0) and nSEL_8x3x8(1) and SEL_8x3x8(0)) or
61 (IN2(0) and SEL_8x3x8(1) and nSEL_8x3x8(0)));
62
63 end mux8x3x8_arc;
```

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity mux8x6x8 is
5
6  port(
7    IN0, IN1, IN2, IN3, IN4, IN5: in std_logic_vector(7 downto 0);
8    SEL_8x6x8: in std_logic_vector(2 downto 0);
9    OUT_8x6x8: out std_logic_vector(7 downto 0)
10   );
11
12 end mux8x6x8;
13
14 architecture mux8x6x8_arc of mux8x6x8 is
15
16 signal nSEL_8x6x8: std_logic_vector(2 downto 0);
17
18 begin
19
20 nSEL_8x6x8(2) <= not SEL_8x6x8(2);
21 nSEL_8x6x8(1) <= not SEL_8x6x8(1);
22 nSEL_8x6x8(0) <= not SEL_8x6x8(0);
23
24 OUT_8x6x8(7) <=
25 ((IN0(7) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
26 (IN1(7) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)) or
27 (IN2(7) and nSEL_8x6x8(2) and SEL_8x6x8(1) and nSEL_8x6x8(0)) or
28 (IN3(7) and nSEL_8x6x8(2) and SEL_8x6x8(1) and SEL_8x6x8(0)) or
29 (IN4(7) and SEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
30 (IN5(7) and SEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)));
31
32 OUT_8x6x8(6) <=
33 ((IN0(6) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
34 (IN1(6) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)) or
35 (IN2(6) and nSEL_8x6x8(2) and SEL_8x6x8(1) and nSEL_8x6x8(0)) or
36 (IN3(6) and nSEL_8x6x8(2) and SEL_8x6x8(1) and SEL_8x6x8(0)) or
37 (IN4(6) and SEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
38 (IN5(6) and SEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)));
39
40 OUT_8x6x8(5) <=
41 ((IN0(5) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
42 (IN1(5) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)) or
43 (IN2(5) and nSEL_8x6x8(2) and SEL_8x6x8(1) and nSEL_8x6x8(0)) or
44 (IN3(5) and nSEL_8x6x8(2) and SEL_8x6x8(1) and SEL_8x6x8(0)) or
45 (IN4(5) and SEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
46 (IN5(5) and SEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)));
47
48 OUT_8x6x8(4) <=
49 ((IN0(4) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
50 (IN1(4) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)) or
51 (IN2(4) and nSEL_8x6x8(2) and SEL_8x6x8(1) and nSEL_8x6x8(0)) or
52 (IN3(4) and nSEL_8x6x8(2) and SEL_8x6x8(1) and SEL_8x6x8(0)) or
53 (IN4(4) and SEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
54 (IN5(4) and SEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)));
55
56 OUT_8x6x8(3) <=
57 ((IN0(3) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
58 (IN1(3) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)) or
59 (IN2(3) and nSEL_8x6x8(2) and SEL_8x6x8(1) and nSEL_8x6x8(0)) or
60 (IN3(3) and nSEL_8x6x8(2) and SEL_8x6x8(1) and SEL_8x6x8(0)) or
61 (IN4(3) and SEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
62 (IN5(3) and SEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)));
63
64 OUT_8x6x8(2) <=
65 ((IN0(2) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
66 (IN1(2) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)) or

```

```
67      (IN2(2) and nSEL_8x6x8(2) and SEL_8x6x8(1) and nSEL_8x6x8(0)) or
68      (IN3(2) and nSEL_8x6x8(2) and SEL_8x6x8(1) and SEL_8x6x8(0)) or
69      (IN4(2) and SEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
70      (IN5(2) and SEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)));
71
72  OUT_8x6x8(1) <=
73  ((IN0(1) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
74  (IN1(1) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)) or
75  (IN2(1) and nSEL_8x6x8(2) and SEL_8x6x8(1) and nSEL_8x6x8(0)) or
76  (IN3(1) and nSEL_8x6x8(2) and SEL_8x6x8(1) and SEL_8x6x8(0)) or
77  (IN4(1) and SEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
78  (IN5(1) and SEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)));
79
80  OUT_8x6x8(0) <=
81  ((IN0(0) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
82  (IN1(0) and nSEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)) or
83  (IN2(0) and nSEL_8x6x8(2) and SEL_8x6x8(1) and nSEL_8x6x8(0)) or
84  (IN3(0) and nSEL_8x6x8(2) and SEL_8x6x8(1) and SEL_8x6x8(0)) or
85  (IN4(0) and SEL_8x6x8(2) and nSEL_8x6x8(1) and nSEL_8x6x8(0)) or
86  (IN5(0) and SEL_8x6x8(2) and nSEL_8x6x8(1) and SEL_8x6x8(0)));
87
88 end mux8x6x8_arc;
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity mux8x16x8 is
5
6 port(
7 INm_8_0, INm_8_1, INm_8_2, INm_8_3,
8 INm_8_4, INm_8_5, INm_8_6, INm_8_7,
9 INm_8_8, INm_8_9, INm_8_10, INm_8_11,
10 INm_8_12, INm_8_13, INm_8_14, INm_8_15 : in std_logic_vector(7 downto 0);
11 ENm_8x16x8: in std_logic;
12 SELm_8x16x8: in std_logic_vector(3 downto 0);
13 OUTm_8x16x8: out std_logic_vector(7 downto 0)
14 );
15
16 end mux8x16x8;
17
18 architecture mux8x16x8_arc of mux8x16x8 is
19
20 component mux16x1 is
21
22 port(
23 INm_16x1: in std_logic_vector(15 downto 0);
24 ENm_16x1: in std_logic;
25 SELm_16x1: in std_logic_vector(3 downto 0);
26 OUTm_16x1: out std_logic
27 );
28
29 end component;
30
31 begin
32
33 mux0: mux16x1 port map(
34 INm_16x1(0) => INm_8_0(0),
35 INm_16x1(1) => INm_8_1(0),
36 INm_16x1(2) => INm_8_2(0),
37 INm_16x1(3) => INm_8_3(0),
38 INm_16x1(4) => INm_8_4(0),
39 INm_16x1(5) => INm_8_5(0),
40 INm_16x1(6) => INm_8_6(0),
41 INm_16x1(7) => INm_8_7(0),
42 INm_16x1(8) => INm_8_8(0),
43 INm_16x1(9) => INm_8_9(0),
44 INm_16x1(10) => INm_8_10(0),
45 INm_16x1(11) => INm_8_11(0),
46 INm_16x1(12) => INm_8_12(0),
47 INm_16x1(13) => INm_8_13(0),
48 INm_16x1(14) => INm_8_14(0),
49 INm_16x1(15) => INm_8_15(0),
50 ENm_16x1 => ENm_8x16x8,
51 SELm_16x1 => SELm_8x16x8,
52 OUTm_16x1 => OUTm_8x16x8(0)
53 );
54
55 mux1: mux16x1 port map(
56 INm_16x1(0) => INm_8_0(1),
57 INm_16x1(1) => INm_8_1(1),
58 INm_16x1(2) => INm_8_2(1),
59 INm_16x1(3) => INm_8_3(1),
60 INm_16x1(4) => INm_8_4(1),
61 INm_16x1(5) => INm_8_5(1),
62 INm_16x1(6) => INm_8_6(1),
63 INm_16x1(7) => INm_8_7(1),
64 INm_16x1(8) => INm_8_8(1),
65 INm_16x1(9) => INm_8_9(1),
66 INm_16x1(10) => INm_8_10(1),
```

```
67  INm_16x1(11) => INm_8_11(1),
68  INm_16x1(12) => INm_8_12(1),
69  INm_16x1(13) => INm_8_13(1),
70  INm_16x1(14) => INm_8_14(1),
71  INm_16x1(15) => INm_8_15(1),
72  ENm_16x1 => ENm_8x16x8,
73  SELm_16x1 => SELm_8x16x8,
74  OUTm_16x1 => OUTm_8x16x8(1)
75 );
76
77 mux2: mux16x1 port map(
78  INm_16x1(0) => INm_8_0(2),
79  INm_16x1(1) => INm_8_1(2),
80  INm_16x1(2) => INm_8_2(2),
81  INm_16x1(3) => INm_8_3(2),
82  INm_16x1(4) => INm_8_4(2),
83  INm_16x1(5) => INm_8_5(2),
84  INm_16x1(6) => INm_8_6(2),
85  INm_16x1(7) => INm_8_7(2),
86  INm_16x1(8) => INm_8_8(2),
87  INm_16x1(9) => INm_8_9(2),
88  INm_16x1(10) => INm_8_10(2),
89  INm_16x1(11) => INm_8_11(2),
90  INm_16x1(12) => INm_8_12(2),
91  INm_16x1(13) => INm_8_13(2),
92  INm_16x1(14) => INm_8_14(2),
93  INm_16x1(15) => INm_8_15(2),
94  ENm_16x1 => ENm_8x16x8,
95  SELm_16x1 => SELm_8x16x8,
96  OUTm_16x1 => OUTm_8x16x8(2)
97 );
98
99 mux3: mux16x1 port map(
100 INm_16x1(0) => INm_8_0(3),
101 INm_16x1(1) => INm_8_1(3),
102 INm_16x1(2) => INm_8_2(3),
103 INm_16x1(3) => INm_8_3(3),
104 INm_16x1(4) => INm_8_4(3),
105 INm_16x1(5) => INm_8_5(3),
106 INm_16x1(6) => INm_8_6(3),
107 INm_16x1(7) => INm_8_7(3),
108 INm_16x1(8) => INm_8_8(3),
109 INm_16x1(9) => INm_8_9(3),
110 INm_16x1(10) => INm_8_10(3),
111 INm_16x1(11) => INm_8_11(3),
112 INm_16x1(12) => INm_8_12(3),
113 INm_16x1(13) => INm_8_13(3),
114 INm_16x1(14) => INm_8_14(3),
115 INm_16x1(15) => INm_8_15(3),
116 ENm_16x1 => ENm_8x16x8,
117 SELm_16x1 => SELm_8x16x8,
118 OUTm_16x1 => OUTm_8x16x8(3)
119 );
120
121 mux4: mux16x1 port map(
122 INm_16x1(0) => INm_8_0(4),
123 INm_16x1(1) => INm_8_1(4),
124 INm_16x1(2) => INm_8_2(4),
125 INm_16x1(3) => INm_8_3(4),
126 INm_16x1(4) => INm_8_4(4),
127 INm_16x1(5) => INm_8_5(4),
128 INm_16x1(6) => INm_8_6(4),
129 INm_16x1(7) => INm_8_7(4),
130 INm_16x1(8) => INm_8_8(4),
131 INm_16x1(9) => INm_8_9(4),
132 INm_16x1(10) => INm_8_10(4),
```

```
133  INm_16x1(11) => INm_8_11(4),
134  INm_16x1(12) => INm_8_12(4),
135  INm_16x1(13) => INm_8_13(4),
136  INm_16x1(14) => INm_8_14(4),
137  INm_16x1(15) => INm_8_15(4),
138  ENm_16x1 => ENm_8x16x8,
139  SELm_16x1 => SELm_8x16x8,
140  OUTm_16x1 => OUTm_8x16x8(4)
141 );
142
143 mux5: mux16x1 port map(
144  INm_16x1(0) => INm_8_0(5),
145  INm_16x1(1) => INm_8_1(5),
146  INm_16x1(2) => INm_8_2(5),
147  INm_16x1(3) => INm_8_3(5),
148  INm_16x1(4) => INm_8_4(5),
149  INm_16x1(5) => INm_8_5(5),
150  INm_16x1(6) => INm_8_6(5),
151  INm_16x1(7) => INm_8_7(5),
152  INm_16x1(8) => INm_8_8(5),
153  INm_16x1(9) => INm_8_9(5),
154  INm_16x1(10) => INm_8_10(5),
155  INm_16x1(11) => INm_8_11(5),
156  INm_16x1(12) => INm_8_12(5),
157  INm_16x1(13) => INm_8_13(5),
158  INm_16x1(14) => INm_8_14(5),
159  INm_16x1(15) => INm_8_15(5),
160  ENm_16x1 => ENm_8x16x8,
161  SELm_16x1 => SELm_8x16x8,
162  OUTm_16x1 => OUTm_8x16x8(5)
163 );
164
165 mux6: mux16x1 port map(
166  INm_16x1(0) => INm_8_0(6),
167  INm_16x1(1) => INm_8_1(6),
168  INm_16x1(2) => INm_8_2(6),
169  INm_16x1(3) => INm_8_3(6),
170  INm_16x1(4) => INm_8_4(6),
171  INm_16x1(5) => INm_8_5(6),
172  INm_16x1(6) => INm_8_6(6),
173  INm_16x1(7) => INm_8_7(6),
174  INm_16x1(8) => INm_8_8(6),
175  INm_16x1(9) => INm_8_9(6),
176  INm_16x1(10) => INm_8_10(6),
177  INm_16x1(11) => INm_8_11(6),
178  INm_16x1(12) => INm_8_12(6),
179  INm_16x1(13) => INm_8_13(6),
180  INm_16x1(14) => INm_8_14(6),
181  INm_16x1(15) => INm_8_15(6),
182  ENm_16x1 => ENm_8x16x8,
183  SELm_16x1 => SELm_8x16x8,
184  OUTm_16x1 => OUTm_8x16x8(6)
185 );
186
187 mux7: mux16x1 port map(
188  INm_16x1(0) => INm_8_0(7),
189  INm_16x1(1) => INm_8_1(7),
190  INm_16x1(2) => INm_8_2(7),
191  INm_16x1(3) => INm_8_3(7),
192  INm_16x1(4) => INm_8_4(7),
193  INm_16x1(5) => INm_8_5(7),
194  INm_16x1(6) => INm_8_6(7),
195  INm_16x1(7) => INm_8_7(7),
196  INm_16x1(8) => INm_8_8(7),
197  INm_16x1(9) => INm_8_9(7),
198  INm_16x1(10) => INm_8_10(7),
```

```
199  INm_16x1(11) => INm_8_11(7),
200  INm_16x1(12) => INm_8_12(7),
201  INm_16x1(13) => INm_8_13(7),
202  INm_16x1(14) => INm_8_14(7),
203  INm_16x1(15) => INm_8_15(7),
204  ENm_16x1 => ENm_8x16x8,
205  SELm_16x1 => SELm_8x16x8,
206  OUTm_16x1 => OUTm_8x16x8(7)
207 );
208
209 end mux8x16x8_arc;
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity mux16x1 is
5
6 port(
7 INm_16x1: in std_logic_vector(15 downto 0);
8 ENm_16x1: in std_logic;
9 SELm_16x1: in std_logic_vector(3 downto 0);
10 OUTm_16x1: out std_logic
11 );
12
13 end mux16x1;
14
15 architecture mux16x1_arc of mux16x1 is
16
17 begin
18
19 OUTm_16x1 <= ENm_16x1 and ((INm_16x1(0) and (not SELm_16x1(3)) and (not SELm_16x1(2)) and
20 (not SELm_16x1(1)) and (not SELm_16x1(0))) or
21 (INm_16x1(1) and (not SELm_16x1(3)) and (not SELm_16x1(2)) and (not SELm_16x1
22 (1)) and (SELm_16x1(0))) or
23 (INm_16x1(2) and (not SELm_16x1(3)) and (not SELm_16x1(2)) and (SELm_16x1(1))
24 and (not SELm_16x1(0))) or
25 (INm_16x1(3) and (not SELm_16x1(3)) and (not SELm_16x1(2)) and (SELm_16x1(1))
26 and (SELm_16x1(0))) or
27 (INm_16x1(4) and (not SELm_16x1(3)) and (SELm_16x1(2)) and (not SELm_16x1(1))
28 and (not SELm_16x1(0))) or
29 (INm_16x1(5) and (not SELm_16x1(3)) and (SELm_16x1(2)) and (not SELm_16x1(1))
30 and (SELm_16x1(0))) or
31 (INm_16x1(6) and (not SELm_16x1(3)) and (SELm_16x1(2)) and (SELm_16x1(1)) and
32 (not SELm_16x1(0))) or
33 (INm_16x1(7) and (not SELm_16x1(3)) and (SELm_16x1(2)) and (SELm_16x1(1)) and
34 (SELm_16x1(0))) or
35 (INm_16x1(8) and (SELm_16x1(3)) and (not SELm_16x1(2)) and (not SELm_16x1(1))
36 and (not SELm_16x1(0))) or
37 (INm_16x1(9) and (SELm_16x1(3)) and (not SELm_16x1(2)) and (not SELm_16x1(1))
38 and (SELm_16x1(0))) or
39 (INm_16x1(10) and (SELm_16x1(3)) and (not SELm_16x1(2)) and (SELm_16x1(1))
40 and (not SELm_16x1(0))) or
41 (INm_16x1(11) and (SELm_16x1(3)) and (not SELm_16x1(2)) and (SELm_16x1(1))
42 and (SELm_16x1(0))) or
43 (INm_16x1(12) and (SELm_16x1(3)) and (SELm_16x1(2)) and (not SELm_16x1(1))
44 and (not SELm_16x1(0))) or
45 (INm_16x1(13) and (SELm_16x1(3)) and (SELm_16x1(2)) and (not SELm_16x1(1))
46 and (SELm_16x1(0))) or
47 (INm_16x1(14) and (SELm_16x1(3)) and (SELm_16x1(2)) and (SELm_16x1(1)) and
48 (not SELm_16x1(0))) or
49 (INm_16x1(15) and (SELm_16x1(3)) and (SELm_16x1(2)) and (SELm_16x1(1)) and
50 (SELm_16x1(0))));
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity mux16x8 is
5
6 port(
7 Mm1, Mm2: in std_logic_vector(7 downto 0);
8 Sm: in std_logic;
9 Xm: out std_logic_vector(7 downto 0)
10 );
11
12 end mux16x8;
13
14 architecture mux16x8_arc of mux16x8 is
15
16 component mux2x1 is
17 port(
18 M1, M2, S: in std_logic;
19 X: out std_logic
20 );
21 end component;
22
23 begin
24
25 mux0: mux2x1 port map(
26 M1 => Mm1(0),
27 M2 => Mm2(0),
28 S => Sm,
29 X => Xm(0)
30 );
31
32 mux1: mux2x1 port map(
33 M1 => Mm1(1),
34 M2 => Mm2(1),
35 S => Sm,
36 X => Xm(1)
37 );
38
39 mux2: mux2x1 port map(
40 M1 => Mm1(2),
41 M2 => Mm2(2),
42 S => Sm,
43 X => Xm(2)
44 );
45
46 mux3: mux2x1 port map(
47 M1 => Mm1(3),
48 M2 => Mm2(3),
49 S => Sm,
50 X => Xm(3)
51 );
52
53 mux4: mux2x1 port map(
54 M1 => Mm1(4),
55 M2 => Mm2(4),
56 S => Sm,
57 X => Xm(4)
58 );
59
60 mux5: mux2x1 port map(
61 M1 => Mm1(5),
62 M2 => Mm2(5),
63 S => Sm,
64 X => Xm(5)
65 );
66
```

```
67  mux6: mux2x1 port map(
68    M1 => Mm1(6),
69    M2 => Mm2(6),
70    S => Sm,
71    X => Xm(6)
72  );
73
74  mux7: mux2x1 port map(
75    M1 => Mm1(7),
76    M2 => Mm2(7),
77    S => Sm,
78    X => Xm(7)
79  );
80
81
82  end mux16x8_arc;
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity mux34x17 is
5
6 port(
7 Mm1, Mm2: in std_logic_vector(16 downto 0);
8 Sm: in std_logic;
9 Xm: out std_logic_vector(16 downto 0)
10 );
11
12 end mux34x17;
13
14 architecture mux34x17_arc of mux34x17 is
15
16 component mux2x1 is
17 port(
18 M1, M2, S: in std_logic;
19 X: out std_logic
20 );
21 end component;
22
23 begin
24
25 mux0: mux2x1 port map(
26 M1 => Mm1(0),
27 M2 => Mm2(0),
28 S => Sm,
29 X => Xm(0)
30 );
31
32 mux1: mux2x1 port map(
33 M1 => Mm1(1),
34 M2 => Mm2(1),
35 S => Sm,
36 X => Xm(1)
37 );
38
39 mux2: mux2x1 port map(
40 M1 => Mm1(2),
41 M2 => Mm2(2),
42 S => Sm,
43 X => Xm(2)
44 );
45
46 mux3: mux2x1 port map(
47 M1 => Mm1(3),
48 M2 => Mm2(3),
49 S => Sm,
50 X => Xm(3)
51 );
52
53 mux4: mux2x1 port map(
54 M1 => Mm1(4),
55 M2 => Mm2(4),
56 S => Sm,
57 X => Xm(4)
58 );
59
60 mux5: mux2x1 port map(
61 M1 => Mm1(5),
62 M2 => Mm2(5),
63 S => Sm,
64 X => Xm(5)
65 );
66
```

```
67  mux6: mux2x1 port map(
68    M1 => Mm1(6),
69    M2 => Mm2(6),
70    S => Sm,
71    X => Xm(6)
72  );
73
74  mux7: mux2x1 port map(
75    M1 => Mm1(7),
76    M2 => Mm2(7),
77    S => Sm,
78    X => Xm(7)
79  );
80
81  mux8: mux2x1 port map(
82    M1 => Mm1(8),
83    M2 => Mm2(8),
84    S => Sm,
85    X => Xm(8)
86  );
87
88  mux9: mux2x1 port map(
89    M1 => Mm1(9),
90    M2 => Mm2(9),
91    S => Sm,
92    X => Xm(9)
93  );
94
95  mux10: mux2x1 port map(
96    M1 => Mm1(10),
97    M2 => Mm2(10),
98    S => Sm,
99    X => Xm(10)
100 );
101
102 mux11: mux2x1 port map(
103   M1 => Mm1(11),
104   M2 => Mm2(11),
105   S => Sm,
106   X => Xm(11)
107 );
108
109 mux12: mux2x1 port map(
110   M1 => Mm1(12),
111   M2 => Mm2(12),
112   S => Sm,
113   X => Xm(12)
114 );
115
116 mux13: mux2x1 port map(
117   M1 => Mm1(13),
118   M2 => Mm2(13),
119   S => Sm,
120   X => Xm(13)
121 );
122
123 mux14: mux2x1 port map(
124   M1 => Mm1(14),
125   M2 => Mm2(14),
126   S => Sm,
127   X => Xm(14)
128 );
129
130 mux15: mux2x1 port map(
131   M1 => Mm1(15),
132   M2 => Mm2(15),
```

```
133     S => Sm,  
134     X => Xm(15)  
135 );  
136  
137 mux16: mux2x1 port map(  
138     M1 => Mm1(16),  
139     M2 => Mm2(16),  
140     S => Sm,  
141     X => Xm(16)  
142 );  
143  
144 end mux34x17_arc;
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity Pilha_LIFO is
5     port (PUSH_p, POP_p, CLK_p: in std_logic;
6             Data_in_p : in std_logic_vector(7 downto 0);
7             Data_out_p : OUT std_logic_vector(7 downto 0));
8 end Pilha_LIFO;
9
10 architecture circuito of Pilha_LIFO is
11     component Cont_updw_4B is
12         port (PRE, CLR: in std_logic_vector(3 downto 0);
13                 UD, CK, EN: IN std_logic ;
14                 S : OUT std_logic_vector(3 downto 0));
15     end component;
16
17     component registers_bank is
18         port (WDATArb: in std_logic_vector(7 downto 0);
19                 WADDRrb, RADDRrb_A, RADDRrb_B: in std_logic_vector(3 downto 0);
20                 WRrb, RDrb_A, RDrb_B, CLKrb, CLRrb: in std_logic;
21                 RDATArb_A, RDATArb_B: out std_logic_vector(7 downto 0));
22     end component;
23
24     signal EN_cont_wr, EN_cont_rd, UD_cont_wr, UD_cont_rd, vazio, cheio_rd, cheio_wr, WR_p,
25 RD_p: std_logic;
26     signal cont_addr_wr, cont_addr_rd: std_logic_vector(3 downto 0);
27
28 begin
29
30     vazio <= NOT(cont_addr_wr(0) OR cont_addr_wr(1) OR cont_addr_wr(2) OR cont_addr_wr(3));
31     cheio_wr <= cont_addr_wr(0) AND cont_addr_wr(1) AND cont_addr_wr(2) AND cont_addr_wr(3);
32     cheio_rd <= cont_addr_rd(0) AND cont_addr_rd(1) AND cont_addr_rd(2) AND cont_addr_rd(3);
33     EN_cont_wr <= (NOT(cheio_wr) AND NOT(vazio) AND NOT(PUSH_p) AND POP_p) OR (NOT(vazio)
34 AND NOT(cheio_rd) AND NOT(PUSH_p) AND POP_p) OR (NOT(cheio_wr) AND NOT(cheio_rd) AND
35 PUSH_p AND NOT(POP_p)) OR (NOT(cheio_wr) AND vazio AND PUSH_p AND NOT(POP_p));
36     UD_cont_wr <= (NOT(cheio_wr) AND NOT(cheio_rd) AND PUSH_p AND NOT(POP_p)) OR (NOT(
37 cheio_wr) AND vazio AND PUSH_p AND NOT(POP_p));
38     EN_cont_rd <= (NOT(vazio) AND NOT(cheio_rd) AND PUSH_p AND NOT(POP_p)) OR (NOT(vazio)
39 AND NOT(PUSH_p) AND POP_p) OR (NOT(cheio_wr) AND cheio_rd AND PUSH_p AND NOT(POP_p));
40     UD_cont_rd <= (NOT(vazio) AND NOT(cheio_rd) AND PUSH_p AND NOT(POP_p)) OR (NOT(cheio_wr)
41 ) AND cheio_rd AND PUSH_p AND NOT(POP_p));
42     WR_p <= (NOT(vazio) AND NOT(cheio_rd) AND PUSH_p AND NOT(POP_p)) OR (NOT(cheio_wr) AND
43 PUSH_p AND NOT(POP_p));
44     RD_p <= NOT(vazio) OR cheio_rd OR '1';
45
46     contador_wr: Cont_updw_4B port map(
47         PRE => "1111",
48         CLR => "1111",
49         UD => UD_cont_wr,
50         CK => CLK_p,
51         EN => EN_cont_wr,
52         S => cont_addr_wr);
53
54     contador_rd: Cont_updw_4B port map(
55         PRE => "1111",
56         CLR => "1111",
57         UD => UD_cont_rd,
58         CK => CLK_p,
59         EN => EN_cont_rd,
60         S => cont_addr_rd);
61
62     banco: registers_bank port map(
63         WDATArb => Data_in_p,
64         WADDRrb => cont_addr_wr,
65         RADDRrb_A => cont_addr_rd,
66         RADDRrb_B => "0000",
67         RDATArb_A => Data_out_p,
68         RDATArb_B => RD_p);
```

```
60      WRrb => WR_p,  
61      RDrb_A => RD_p,  
62      RDrb_B => '0',  
63      CLKrb => CLK_p,  
64      CLRrb => '1',  
65      RDATArb_A => Data_out_p);  
66  
67  
68 end circuito;
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity eight_bit_register is
5
6 port(
7 De: in std_logic_vector(7 downto 0);
8 CLKe, Se, RSe, ENe: in std_logic;
9 Qe: out std_logic_vector(7 downto 0)
10 );
11
12 end eight_bit_register;
13
14 architecture eight_bit_register_arc of eight_bit_register is
15
16 component FFD is
17 port(
18
19 D, CLK, S, RS: in std_logic;
20 Q: out std_logic
21 );
22
23 end component;
24
25 component mux16x8 is
26
27 port(
28 Mm1, Mm2: in std_logic_vector(7 downto 0);
29 Sm: in std_logic;
30 Xm: out std_logic_vector(7 downto 0)
31 );
32
33 end component;
34
35 signal Dem, Qer: std_logic_vector(7 downto 0);
36
37 begin
38
39 mult: mux16x8 port map(
40     Mm1 => Qer,
41     Mm2 => De,
42     Sm => ENe,
43     Xm => Dem
44 );
45
46 reg0: FFD port map(
47     D => Dem(0),
48     CLK => CLKe,
49     S => Se,
50     RS => RSe,
51     Q => Qer(0)
52 );
53
54 reg1: FFD port map(
55     D => Dem(1),
56     CLK => CLKe,
57     S => Se,
58     RS => RSe,
59     Q => Qer(1)
60 );
61
62 reg2: FFD port map(
63     D => Dem(2),
64     CLK => CLKe,
65     S => Se,
66     RS => RSe,
```

```
67      Q => Qer(2)
68  );
69
70  reg3: FFD port map(
71      D => Dem(3),
72      CLK => CLKE,
73      S => Se,
74      RS => RSe,
75      Q => Qer(3)
76  );
77
78  reg4: FFD port map(
79      D => Dem(4),
80      CLK => CLKE,
81      S => Se,
82      RS => RSe,
83      Q => Qer(4)
84  );
85
86  reg5: FFD port map(
87      D => Dem(5),
88      CLK => CLKE,
89      S => Se,
90      RS => RSe,
91      Q => Qer(5)
92  );
93
94  reg6: FFD port map(
95      D => Dem(6),
96      CLK => CLKE,
97      S => Se,
98      RS => RSe,
99      Q => Qer(6)
100 );
101
102 reg7: FFD port map(
103     D => Dem(7),
104     CLK => CLKE,
105     S => Se,
106     RS => RSe,
107     Q => Qer(7)
108 );
109
110 Qe <= Qer;
111
112 end eight_bit_register_arc;
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity ULA is
5      port(A_ula: in std_logic_vector(7 downto 0);
6            B_ula: in std_logic_vector(7 downto 0);
7            Se_ula: in std_logic_vector(3 downto 0);
8            Out_ula: out std_logic_vector(7 downto 0);
9            led_ovf_sum: out std_logic;
10           led_ovf_mult: out std_logic;
11           led_ovf_inc: out std_logic;
12           jump_eq: out std_logic;
13           jump_hi: out std_logic;
14           jump_lo: out std_logic);
15
16 end ULA;
17
18
19 architecture arrangement of ULA is
20
21 component somador8bits is
22     port(A: in std_logic_vector(7 downto 0);
23           B: in std_logic_vector(7 downto 0 );
24           Ci: in std_logic;
25           Co: out std_logic;
26           S_MSB: out std_logic_vector(7 downto 0)
27           );
28
29 end component;
30
31 component substrator8bits is
32     port(A_sub: in std_logic_vector(7 downto 0);
33           B_sub: in std_logic_vector(7 downto 0 );
34           En_sub: in std_logic;
35           S_sub: out std_logic_vector(7 downto 0)
36           );
37
38
39 end component;
40
41 component mux8x16x8 is
42     port(
43             INm_8_0, INm_8_1, INm_8_2, INm_8_3,
44             INm_8_4, INm_8_5, INm_8_6, INm_8_7,
45             INm_8_8, INm_8_9, INm_8_10, INm_8_11,
46             INm_8_12, INm_8_13, INm_8_14, INm_8_15 : in std_logic_vector(7 downto 0);
47             ENm_8x16x8: in std_logic;
48             SELm_8x16x8: in std_logic_vector(3 downto 0);
49             OUTm_8x16x8: out std_logic_vector(7 downto 0)
50           );
51
52 end component;
53
54 component decremento is
55     port(A_dec: in std_logic_vector(7 downto 0);
56           S_dec: out std_logic_vector(7 downto 0)
57           );
58 end component;
59
60 component incremento is
61     port(A_inc: in std_logic_vector(7 downto 0);
62           Co_inc: out std_logic;
63           S_inc: out std_logic_vector(7 downto 0)
64           );
65 end component;
```

```
67 component SHR is
68   port(A_shr:  in std_logic_vector(7 downto 0);
69     S_shr:  out std_logic_vector(7 downto 0)
70   );
71 end component;
72
73 component SHL is
74   port(A_shl:  in std_logic_vector(7 downto 0);
75     S_shl:  out std_logic_vector(7 downto 0)
76   );
77 end component;
78
79 component comparador is
80   port(A,B:  in std_logic_vector(7 downto 0);
81     eq,bt,lt:  out std_logic);
82
83 end component;
84
85 component multiplicador is
86   port (A_M:  in std_logic_vector(7 downto 0);
87     B_M:  in std_logic_vector(7 downto 0);
88     S_M:  out std_logic_vector(15 downto 0));
89
90 end component;
91
92
93 signal car_out, signal_carry_inc, comp_sum_flag, comp_mult_flag: std_logic;
94 signal signal_out_sum, signal_out_sub, signal_out_mult_P1, signal_out_mult_P2,
95 signal_out_inc, signal_out_dec,
96 signal_and, signal_not, signal_xor, signal_or, signal_out_shr, signal_out_shl:
97 std_logic_vector(7 downto 0);
98 signal out_comp: std_logic_vector(7 downto 0);
99 signal signal_out_mult: std_logic_vector(15 downto 0);
100
101 begin
102 SOMADOR1: somador8bits port map(
103   A => A_ula,
104   B => B_ula,
105   Ci => '0',
106   Co => car_out,
107   S_MSB => signal_out_sum);
108
109 SUBTRATOR1: subtrator8bits port map(
110   A_sub => A_ula,
111   B_sub => B_ula,
112   En_sub => '1',
113   S_sub => signal_out_sub);
114
115 COMP: comparador port map(
116   A => A_ula,
117   B => B_ula,
118   eq => out_comp(2),
119   bt => out_comp(1),
120   lt => out_comp(0));
121
122 out_comp(7) <= '0';
123 out_comp(6) <= '0';
124 out_comp(5) <= '0';
125 out_comp(4) <= '0';
126 out_comp(3) <= '0';
127 jump_eq <= out_comp(2);
128 jump_hi <= out_comp(1);
129 jump_lo <= out_comp(0);
130
131 MULT: multiplicador port map(
```

```
131      A_M => A_ula,
132      B_M => B_ula,
133      S_M => signal_out_mult);
134
135      signal_out_mult_P1 <= signal_out_mult(15 downto 8);
136      signal_out_mult_P2 <= signal_out_mult(7 downto 0);
137
138  INC: incremento port map(
139      A_inc => A_ula,
140      Co_inc => signal_carry_inc,
141      S_inc => signal_out_inc);
142
143  DEC: decremento port map(
144      A_dec => A_ula,
145      S_dec => signal_out_dec);
146
147
148      signal_and <= A_ula and B_ula;
149      signal_or <= A_ula or B_ula;
150      signal_xor <= A_ula xor B_ula;
151      signal_not <= not A_ula;
152
153  SHR01: SHR port map(
154      A_shr => A_ula,
155      S_shr => signal_out_shr);
156
157  SHL01: SHL port map(
158      A_shl => A_ula,
159      S_shl => signal_out_shl);
160
161      led_ovf_mult <= signal_out_mult_P1(7) or signal_out_mult_P1(6) or signal_out_mult_P1(5) or
162      signal_out_mult_P1(4) or signal_out_mult_P1(3) or signal_out_mult_P1(2) or
163      signal_out_mult_P1(1) or signal_out_mult_P1(0);
164      led_ovf_sum <= car_out;
165      led_ovf_inc <= signal_carry_inc;
166
167  MUXULA01: mux8x16x8 port map(
168      INm_8_0 => signal_out_sum,
169      INm_8_1 => signal_out_sub,
170      INm_8_2 => out_comp,
171      INm_8_3 => signal_out_mult_P2,
172      INm_8_4 => signal_out_inc,
173      INm_8_5 => signal_out_dec,
174      INm_8_6 => signal_and,
175      INm_8_7 => signal_or,
176      INm_8_8 => signal_xor,
177      INm_8_9 => signal_not,
178      INm_8_10 => signal_out_shl,
179      INm_8_11 => signal_out_shr,
180      INm_8_12 => "00000000",
181      INm_8_13 => "00000000",
182      INm_8_14 => "00000000",
183      INm_8_15 => "00000000",
184      ENm_8x16x8 => '1',
185      SELm_8x16x8 => Se_ula,
186      OUTm_8x16x8 => out_ula);
187
188
189
190 end arrangement;
```

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity multiplicador is
5      port (A_M: in std_logic_vector(7 downto 0);
6             B_M: in std_logic_vector(7 downto 0);
7             S_M: out std_logic_vector(15 downto 0));
8
9  end multiplicador;
10
11 architecture funcionamento of multiplicador is
12
13     component somador16std_logics is
14         port(A:  in std_logic_vector(15 downto 0);
15              B:  in std_logic_vector(15 downto 0 );
16              Ci: in std_logic;
17              Co: out std_logic;
18              S:   out std_logic_vector(15 downto 0));
19
20     end component;
21
22
23     signal mul0, mul1,mul2, mul3,mul4,mul5,mul6, mul7: std_logic_vector(15 downto 0);
24     signal sum0, sum1 ,sum2, sum3 , sum4, sum5 : std_logic_vector(15 downto 0);
25     begin
26
27         mul0(15 downto 8)<="00000000";
28         mul0(7)<=B_M(0) and A_M(7);
29         mul0(6)<=B_M(0) and A_M(6);
30         mul0(5)<=B_M(0) and A_M(5);
31         mul0(4)<=B_M(0) and A_M(4);
32         mul0(3)<=B_M(0) and A_M(3);
33         mul0(2)<=B_M(0) and A_M(2);
34         mul0(1)<=B_M(0) and A_M(1);
35         mul0(0)<=B_M(0) and A_M(0);
36
37         -----
38         mul1(15 downto 9)<="00000000";
39         mul1(8)<=B_M(1) and A_M(7);
40         mul1(7)<=B_M(1) and A_M(6);
41         mul1(6)<=B_M(1) and A_M(5);
42         mul1(5)<=B_M(1) and A_M(4);
43         mul1(4)<=B_M(1) and A_M(3);
44         mul1(3)<=B_M(1) and A_M(2);
45         mul1(2)<=B_M(1) and A_M(1);
46         mul1(1)<=B_M(1) and A_M(0);
47         mul1(0)<='0';
48
49         -----
50
51         mul2(15 downto 10)<="000000";
52         mul2(9)<=B_M(2) and A_M(7);
53         mul2(8)<=B_M(2) and A_M(6);
54         mul2(7)<=B_M(2) and A_M(5);
55         mul2(6)<=B_M(2) and A_M(4);
56         mul2(5)<=B_M(2) and A_M(3);
57         mul2(4)<=B_M(2) and A_M(2);
58         mul2(3)<=B_M(2) and A_M(1);
59         mul2(2)<=B_M(2) and A_M(0);
60         mul2(1 downto 0)<="00";
61
62         -----
63
64         mul3(15 downto 11)<="00000";
65         mul3(10)<=B_M(3) and A_M(7);
66         mul3(9)<=B_M(3) and A_M(6);

```

```

67      mul3(8)<=B_M(3) and A_M(5);
68      mul3(7)<=B_M(3) and A_M(4);
69      mul3(6)<=B_M(3) and A_M(3);
70      mul3(5)<=B_M(3) and A_M(2);
71      mul3(4)<=B_M(3) and A_M(1);
72      mul3(3)<=B_M(3) and A_M(0);
73      mul3(2 downto 0)<="000";
74 -----
75      mul4(15 downto 12)<="0000";
76      mul4(11)<=B_M(4) and A_M(7);
77      mul4(10)<=B_M(4) and A_M(6);
78      mul4(9)<=B_M(4) and A_M(5);
79      mul4(8)<=B_M(4) and A_M(4);
80      mul4(7)<=B_M(4) and A_M(3);
81      mul4(6)<=B_M(4) and A_M(2);
82      mul4(5)<=B_M(4) and A_M(1);
83      mul4(4)<=B_M(4) and A_M(0);
84      mul4(3 downto 0)<="0000";
85 -----
86      mul5(15 downto 13)<="000";
87      mul5(12)<=B_M(5) and A_M(7);
88      mul5(11)<=B_M(5) and A_M(6);
89      mul5(10)<=B_M(5) and A_M(5);
90      mul5(9)<=B_M(5) and A_M(4);
91      mul5(8)<=B_M(5) and A_M(3);
92      mul5(7)<=B_M(5) and A_M(2);
93      mul5(6)<=B_M(5) and A_M(1);
94      mul5(5)<=B_M(5) and A_M(0);
95      mul5(4 downto 0)<="00000";
96 -----
97      mul6(15 downto 14)<="00";
98      mul6(13)<=B_M(6) and A_M(7);
99      mul6(12)<=B_M(6) and A_M(6);
100     mul6(11)<=B_M(6) and A_M(5);
101     mul6(10)<=B_M(6) and A_M(4);
102     mul6(9)<=B_M(6) and A_M(3);
103     mul6(8)<=B_M(6) and A_M(2);
104     mul6(7)<=B_M(6) and A_M(1);
105     mul6(6)<=B_M(6) and A_M(0);
106     mul6(5 downto 0)<="000000";
107 -----
108    mul7(15)<='0';
109    mul7(14)<=B_M(7) and A_M(7);
110    mul7(13)<=B_M(7) and A_M(6);
111    mul7(12)<=B_M(7) and A_M(5);
112    mul7(11)<=B_M(7) and A_M(4);
113    mul7(10)<=B_M(7) and A_M(3);
114    mul7(9)<=B_M(7) and A_M(2);
115    mul7(8)<=B_M(7) and A_M(1);
116    mul7(7)<=B_M(7) and A_M(0);
117    mul7(6 downto 0)<="0000000";
118
119
120    SOMADOR0: somador16std_logics port map(
121
122        A => mul0,
123        B => mul1,
124        Ci => '0',
125        S => sum0);
126
127
128    SOMADOR1: somador16std_logics port map(
129
130        A => sum0,
131        B => mul2,
132        Ci => '0',

```

```
133         S => sum1);  
134  
135  
136     SOMADOR2: somador16std_logics port map(  
137  
138         A => sum1,  
139         B => mul3,  
140         Ci => '0',  
141         S => sum2);  
142  
143  
144     SOMADOR3: somador16std_logics port map(  
145  
146         A => sum2,  
147         B => mul4,  
148         Ci => '0',  
149         S => sum3);  
150  
151  
152     SOMADOR4: somador16std_logics port map(  
153  
154         A => sum3,  
155         B => mul5,  
156         Ci => '0',  
157         S => sum4);  
158  
159  
160     SOMADOR5: somador16std_logics port map(  
161  
162         A => sum4,  
163         B => mul6,  
164         Ci => '0',  
165         S => sum5);  
166  
167  
168     SOMADOR6: somador16std_logics port map(  
169  
170         A => sum5,  
171         B => mul7,  
172         Ci => '0',  
173         S => S_M);  
174  
175  
176  
177  
178  
179     end funcionamento;  
180  
181
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity SHL is
5     port(A_shl:  in std_logic_vector(7 downto 0);
6          S_shl:  out std_logic_vector(7 downto 0)
7          );
8 end SHL;
9
10 architecture arrangement of SHL is
11 begin
12
13     S_shl(7) <= A_shl(6);
14     S_shl(6) <= A_shl(5);
15     S_shl(5) <= A_shl(4);
16     S_shl(4) <= A_shl(3);
17     S_shl(3) <= A_shl(2);
18     S_shl(2) <= A_shl(1);
19     S_shl(1) <= A_shl(0);
20     S_shl(0) <= '0';
21
22
23 end arrangement;
24
25
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity SHR is
5     port(A_shr:  in std_logic_vector(7 downto 0);
6          S_shr:  out std_logic_vector(7 downto 0)
7          );
8 end SHR;
9
10 architecture arrangement of SHR is
11
12 begin
13
14     S_shr(7) <= '0';
15     S_shr(6) <= A_shr(7);
16     S_shr(5) <= A_shr(6);
17     S_shr(4) <= A_shr(5);
18     S_shr(3) <= A_shr(4);
19     S_shr(2) <= A_shr(3);
20     S_shr(1) <= A_shr(2);
21     S_shr(0) <= A_shr(1);
22
23
24 end arrangement;
25
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity comparador is
5     port(A,B: in std_logic_vector(7 downto 0);
6          eq,bt,lt: out std_logic);
7
8 end comparador;
9
10 architecture funcionamento of comparador is
11
12     signal eqout,btout: std_logic;
13     signal ab: std_logic_vector(7 downto 0);
14
15 begin
16
17     ab <= A xnor B;
18
19
20     eqout <= ab(0) and ab(1) and ab(2) and ab(3) and ab(4) and ab(5) and ab(6) and ab(7);
21     btout <= (not B(7) and A(7)) or (ab(7) and not B(6) and A(6)) or (ab(7) and ab(6) and
22     not B(5) and A(5)) or (ab(7) and ab(6) and ab(5) and not B(4) and A(4)) or (ab(7) and ab(6)
23     ) and ab(5) and ab(4) and not B(3) and A(3)) or (ab(7) and ab(6) and ab(5) and ab(4) and
24     ab(3) and not B(2) and A(2)) or (ab(7) and ab(6) and ab(5) and ab(4) and ab(3) and ab(2)
25     and not B(1) and A(1)) or (ab(7) and ab(6) and ab(5) and ab(4) and ab(3) and ab(2) and ab(
26     1) and not B(0) and A(0));
27     lt <= eqout nor btout;
28
29     eq <= eqout;
30     bt <= btout;
31
32 end funcionamento;
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity somador is
5     port (a: in std_logic;
6             b: in std_logic;
7             ci: in std_logic;
8             co: out std_logic;
9             s: out std_logic );
10
11 end somador;
12
13
14 architecture funcionamento of somador is
15 begin
16
17     co <= (b and ci) or (a and ci) or (a and b);
18     s <= a xor b xor ci;
19
20 end funcionamento;
21
22
23
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity somador16std_logics is
5      port(A:  in std_logic_vector(15 downto 0);
6            B:  in std_logic_vector(15 downto 0 );
7            Ci: in std_logic;
8            Co: out std_logic;
9            S:  out std_logic_vector(15 downto 0));
10
11 end somador16std_logics;
12
13 architecture arrangement of somador16std_logics is
14
15     component somador is
16         port (a: in std_logic;
17                b: in std_logic;
18                ci: in std_logic;
19                co: out std_logic;
20                s: out std_logic);
21
22     end component;
23
24     signal carry_out: std_logic_vector(0 to 14) ;
25
26 begin
27
28     SOMADOR01: somador port map(
29         a => A(0),
30         b => B(0),
31         ci => Ci,
32         co => carry_out(0),
33         s => S(0));
34
35     SOMADOR02: somador port map(
36         a => A(1),
37         b => B(1),
38         ci => carry_out(0),
39         co => carry_out(1),
40         s => S(1));
41
42
43     SOMADOR03: somador port map(
44         a => A(2),
45         b => B(2),
46         ci => carry_out(1),
47         co => carry_out(2),
48         s => S(2));
49
50
51     SOMADOR04: somador port map(
52         a => A(3),
53         b => B(3),
54         ci => carry_out(2),
55         co => carry_out(3),
56         s => S(3));
57
58     SOMADOR05: somador port map(
59         a => A(4),
60         b => B(4),
61         ci => carry_out(3),
62         co => carry_out(4),
63         s => S(4));
64
65     SOMADOR06: somador port map(
66         a => A(5),
```

```
67      b => B(5),
68      ci => carry_out(4),
69      co => carry_out(5),
70      s => S(5));
71
72  SOMADOR07: somador port map(
73      a => A(6),
74      b => B(6),
75      ci => carry_out(5),
76      co => carry_out(6),
77      s => S(6));
78
79  SOMADOR08: somador port map(
80      a => A(7),
81      b => B(7),
82      ci => carry_out(6),
83      co => carry_out(7),
84      s => S(7));
85
86  SOMADOR09: somador port map(
87      a => A(8),
88      b => B(8),
89      ci => carry_out(7),
90      co => carry_out(8),
91      s => S(8));
92
93  SOMADOR10: somador port map(
94      a => A(9),
95      b => B(9),
96      ci => carry_out(8),
97      co => carry_out(9),
98      s => S(9));
99
100 SOMADOR11: somador port map(
101     a => A(10),
102     b => B(10),
103     ci => carry_out(9),
104     co => carry_out(10),
105     s => S(10));
106
107 SOMADOR12: somador port map(
108     a => A(11),
109     b => B(11),
110     ci => carry_out(10),
111     co => carry_out(11),
112     s => S(11));
113
114 SOMADOR13: somador port map(
115     a => A(12),
116     b => B(12),
117     ci => carry_out(11),
118     co => carry_out(12),
119     s => S(12));
120
121 SOMADOR14: somador port map(
122     a => A(13),
123     b => B(13),
124     ci => carry_out(12),
125     co => carry_out(13),
126     s => S(13));
127
128 SOMADOR15: somador port map(
129     a => A(14),
130     b => B(14),
131     ci => carry_out(13),
132     co => carry_out(14),
```

```
133      s => S(14)) ;  
134  
135  SOMADOR16: somador port map(  
136      a => A(15),  
137      b => B(15),  
138      ci => carry_out(14),  
139      co => Co,  
140      s => S(15)) ;  
141  
142  
143 end arrangement;  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity somador8bits is
5      port(A:  in std_logic_vector(7 downto 0);
6            B:  in std_logic_vector(7 downto 0 );
7            Ci: in std_logic;
8            Co: out std_logic;
9            S_MSB: out std_logic_vector(7 downto 0)
10           );
11
12 end somador8bits;
13
14 architecture arrangement of somador8bits is
15
16     component somador is
17         port (a: in std_logic;
18                b: in std_logic;
19                ci: in std_logic;
20                co: out std_logic;
21                s: out std_logic);
22
23 end component;
24
25     signal carry_out: std_logic_vector(0 to 6) ;
26
27 begin
28
29     SOMADOR01: somador port map(
30         a => A(0),
31         b => B(0),
32         ci => Ci,
33         co => carry_out(0),
34         s => S_MSB(0));
35
36     SOMADOR02: somador port map(
37         a => A(1),
38         b => B(1),
39         ci => carry_out(0),
40         co => carry_out(1),
41         s => S_MSB(1));
42
43
44     SOMADOR03: somador port map(
45         a => A(2),
46         b => B(2),
47         ci => carry_out(1),
48         co => carry_out(2),
49         s => S_MSB(2));
50
51
52     SOMADOR04: somador port map(
53         a => A(3),
54         b => B(3),
55         ci => carry_out(2),
56         co => carry_out(3),
57         s => S_MSB(3));
58
59     SOMADOR05: somador port map(
60         a => A(4),
61         b => B(4),
62         ci => carry_out(3),
63         co => carry_out(4),
64         s => S_MSB(4));
65
66     SOMADOR06: somador port map(
```

```
67      a => A(5),
68      b => B(5),
69      ci => carry_out(4),
70      co => carry_out(5),
71      s => S_MSB(5));
72
73      SOMADOR07: somador port map(
74          a => A(6),
75          b => B(6),
76          ci => carry_out(5),
77          co => carry_out(6),
78          s => S_MSB(6));
79
80      SOMADOR08: somador port map(
81          a => A(7),
82          b => B(7),
83          ci => carry_out(6),
84          co => Co,
85          s => S_MSB(7));
86
87
88
89  end arrangement;
90
91
92
93
94
95
96
97
98
99
100
101
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity subtrator8bits is
5      port(A_sub:  in std_logic_vector(7 downto 0);
6            B_sub:  in std_logic_vector(7 downto 0 );
7            En_sub: in std_logic;
8            S_sub:  out std_logic_vector(7 downto 0)
9            );
10
11
12 end subtrator8bits;
13
14 architecture arrangement of subtrator8bits is
15
16     component somador8bits is
17         port( A:  in std_logic_vector(7 downto 0);
18               B:  in std_logic_vector(7 downto 0 );
19               Ci: in std_logic;
20               Co: out std_logic;
21               S_MSB: out std_logic_vector(7 downto 0));
22
23 end component;
24
25     component mux2x1_s is
26         port(A: in std_logic_vector(7 downto 0);
27               B: in std_logic_vector(7 downto 0 );
28               Se: in std_logic ;
29               Sa: out std_logic_vector(7 downto 0));
30
31 end component;
32
33
34     signal not_B, out_mux: std_logic_vector(7 downto 0);
35     signal co_sub: std_logic;
36
37
38 begin
39
40
41     MUX01: mux2x1_s port map(
42         A => B_sub,
43         B => not_B,
44         Se => En_sub,
45         Sa => out_mux);
46
47
48
49     SUBTRATOR01: somador8bits port map(
50         A => A_sub,
51         B => out_mux,
52         Ci => En_sub, --talvez colocar not en_sub por causa da flag--
53         Co => co_sub,
54         S_MSB => S_sub);
55
56
57
58 end arrangement;
59
60
61
62
63
64
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity decremento is
5     port(A_dec:  in std_logic_vector(7 downto 0);
6          S_dec:  out std_logic_vector(7 downto 0)
7          );
8 end decremento;
9
10 architecture funcionamento of decremento is
11
12     component subtrator8bits is
13         port(A_sub:  in std_logic_vector(7 downto 0);
14             B_sub:  in std_logic_vector(7 downto 0 );
15             En_sub: in std_logic;
16             S_sub:  out std_logic_vector(7 downto 0)
17             );
18
19
20     end component;
21
22
23 begin
24
25     SUBTRATOR01: subtrator8bits port map(
26         A_sub => A_dec,
27         B_sub =>"00000001",
28         En_sub => '1',
29         S_sub => S_dec);
30
31 end funcionamento;
32
33
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity incremento is
5      port(A_inc:  in std_logic_vector(7 downto 0);
6            Co_inc: out std_logic;
7            S_inc:  out std_logic_vector(7 downto 0)
8            );
9  end incremento;
10
11 architecture funcionamento of incremento is
12
13     component somador8bits is
14         port( A:  in std_logic_vector(7 downto 0);
15               B:  in std_logic_vector(7 downto 0 );
16               Ci: in std_logic;
17               Co: out std_logic;
18               S_MSB:  out std_logic_vector(7 downto 0));
19
20     end component;
21
22
23 begin
24
25     SOMADOR01: somador8bits port map(
26             A => A_inc,
27             B => "00000001",
28             Ci => '0',
29             Co => Co_inc,
30             S_MSB => S_inc);
31
32 end funcionamento;
33
34
```