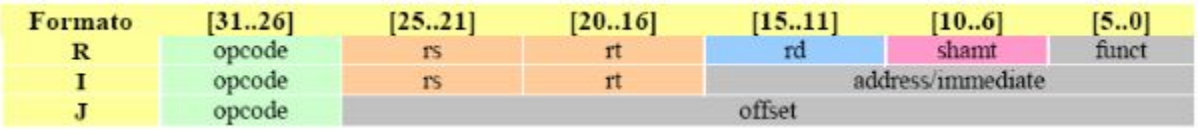
Data Path



Formato	[31..26]	[25..21]	[20..16]	[15..11]	[10..6]	[5..0]
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	address/immediate		
J	opcode	offset				

Assembly	Opcode	rs	rt	rd	shamt	funct	Comportamento	Ciclos
`add rd, rs, rt`	0x0	rs	rt	rd	0x0	0x20	Rd ← rs + rt	3
`and rd, rs, rt`	0x0	rs	rt	rd	0x0	0x24	Rd ← rs & rt	3
`div rs, rt`	0x0	rs	rt	-	0x0	0x1a	**(Ver divisão)**	**3**
`mult rs,rt`	0x0	rs	rt	-	0x0	0x18	**(Ver multiplicação)**	**3**
`jr rs`	0x0	rs	-	-	0x0	0x8	PC ← rs	3
`mfhi rd`	0x0	-	-	rd	0x0	0x10	rd ← hi	2
`mflo rd`	0x0	-	-	rd	0x0	0x12	rd ← lo	2
`sll rd, rt, shamt`	0x0	-	rt	rd	shamt	0x0	rd ← rt << shamt	4
`sllv rd, rs, rt`	0x0	rs	rt	rd	0x0	0x4	rd ← rs << rt	4
`slt rd, rs, rt`	0x0	rs	rt	rd	0x0	0x2a	rd ← (rs < rt) ?1 :0	3
`srl rd, rt, shamt`	0x0	-	rt	rd	shamt	0x2	rd ← rt >> shamt	4
`sub rd, rs, rt`	0x0	rs	rt	rd	0x0	0x22	rd ← rs - rt	3
`Rte`	0x0	-	-	-	0x0	0x13	PC EPC	2
`Push rt`	0x0	-	rt	-	0x0	0x5	**SP <- rt**	3
`Pop rt`	0x0	-	rt	-	0x0	0x6	**(Ver push e pop)**	4

Tabela 2. Instruções Formato I

Assembly	Opcode	rs	rt	End/Imediato	Comportamento	Ciclos
`addi rt, rs, imediato`	0x8	rs	rt	imediato	Rt ← rs + imediato**	3

`beq rs,rt, offset`	0x4	rs	rt	offset	Desvia se rs == rt (ver desvios)	3
`bne rs,rt, offset`	0x5	rs	rt	offset	Desvia se rs != rt (ver desvios)	3
`ble rs,rt,offset`	0x6	rs	rt	offset	Desvia se rs <= rt (ver desvios)	3
`bgt rs,rt,offset`	0x7	rs	rt	offset	Desvia se rs > rt (ver desvios)	3
`lb rt, offset(rs)`	0x20	rs	rt	offset	Rt ← byte Mem[offset + rs]	5
`lh rt, offset(rs)`	0x21	rs	rt	offset	Rt ← halfword Mem[offset + rs]	5
`lui rt, imediato`	0xf	-	rt	imediato	Rt ← imediato << 16	3
`lw rt, offset(rs)`	0x23	rs	rt	end	Rt ← Mem[offset + rs]	4
`sb rt, offset(rs)`	0x28	rs	rt	offset	Mem[offset +rs] byte[rt] (ver stores)	6
`slti rt, rs, imediato`	0xa	rs	rt	imediato	Rt ← (rs < imediato) ?1 :0	3
`sw rt, offset(rs)`	0x2b	rs	rt	offset	Mem[offset +rs] ← rt	3

***** o valor de 'imediato' deve ser estendido para 32 bits, estendendo seu sinal (bit mais significativo da constante).***

Tabela 3. Instruções Formato J

Assembly	Opcode	Offset	Comportamento	Ciclos
`j offset`	0x2	offset	(ver desvios)	2
`jal offset`	0x3	offset	(ver desvios)	2

Nome	Efeito quando 0	Efeito quando 1
RegDst	Número do registrador destino para escrita vem de rt	Número do registrador destino para escrita vem de rd
RegWrite	Nada	Registrador da entrada "Write register" recebe valor da entrada "Write data"
ALUSrcA	1° operando da ALU vem do PC	1° operando da ALU vem do registrador A

IRWrite	Nada	Saída da memória é escrita em IR
MemRead	Nada	Dado da memória relativo ao endereço especificado é colocado na saída
MemWrite	Nada	Dado da memória do endereço especificado é substituido pelo que tem no “Write data”
MemTo Reg	Valor escrito na entrada “Write data” dos registradores vem da ALUOut	Valor escrito na entrada “Write data” vem de MDR
PCWrite	Nada	Escrita em PC, controlado por PCSource
PCWriteCond	Nada	Escrita em PC se saída Zero de ALU ativo
IorD	PC fornece endereço para a memória	ALUOut fornece endereço para a memória

1Nome	Valor	Efeito
ALUOp	000	ALU faz soma
	001	ALU faz subtração
	010	Campo funct determina operação
	011	ALU faz comparação
	100	ALU faz ANDlógico
	101	ALU faz ORlógico

ALUSrcB	00	2° Operando da ALU vem de B
	01	2° Operando da ALU é a constante 4
	10	2° Operando é o sinal estendido, 16 bits menos significantes de IR
	11	2° Operando é o sinal estendido, 16 bits menos significantes de IR, deslocados de 2 bits para a esquerda
PCSource	00	Saída de ALU (PC + 4) é enviado ao PC para escrita
	01	ALUOut (endereço destino do branch) é enviado ao PC para escrita
	10	Endereço destino do jump (26 bits menos significativos do IR deslocados de 2 bits para a esquerda concatenados com PC+4[31:28]) é enviado ao PC para escrita

Especificação da instruções: http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html

2.7 Push e Pop

As instruções push e pop servem respectivamente para armazenar registradores na pilha da memória e restaurar valores de registradores que estão na pilha. A utilização destas instruções implica em atualizar também o valor do registrador SP (registrador 29). A pilha cresce de endereços maiores para menores como explicado anteriormente. Portanto além de armazenar (ou ler) um valor da memória, o registrador SP deve ser atualizado.

Exemplo: push rt – Supondo que o registrador SP aponte para o endereço 227(decimal), ele deve ser decrementado em 4 (223) e o conteúdo de rt deve ser armazenado nesta posição de memória(223).

Exemplo: pop rt – Supondo que o registrador SP aponte para o endereço 223(decimal),o conteúdo desta posição de memória deve ser armazenado no registrador rt e o registrador SP deve ser incrementado em 4 (227) .

2.3 - Divisão

A divisão deve ser implementada como um componente à parte, o **Divisor**, não sendo recomendada a reutilização da ALU principal do processador para efetuar qualquer operação referente a esta função. Este componente deve realizar corretamente a divisão de dois números de 32 bits, com sinal, resultando em um número de 32 bits, também com sinal.

O Divisor deve ter também um sinal de saída que indica se houve divisão por zero. Este sinal deve servir para lançar a exceção de divisão por zero (ver exceções).

Instrução	Descrição
div rs, rt	Realiza a divisão entre rs e rt, armazenando internamente (em dois registradores próprios do divisor) o resultado.
mfhi rd	Copia a parte alta do resultado da divisão - o resto (32 bits mais significativos) para o registrador rd.
mflo rd	Copia a parte baixa do resultado da divisão - quociente (32 bits menos significativos) para o registrador rd.

Importante: Note que o resultado da divisão inteira é armazenado em dois registradores. O registrador Hi deve conter o resto e o registrador Lo o quociente.

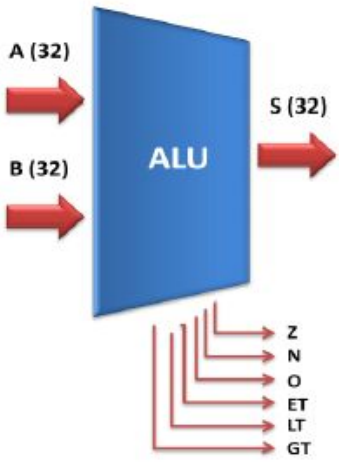
2.4 - Multiplicação

A multiplicação deve ser implementada como um componente à parte, o **Multiplicador**, não sendo recomendada a reutilização da ALU principal do processador para efetuar qualquer operação referente a esta função. Este componente realizará corretamente a multiplicação de dois números de 32 bits, com sinal, resultando em um número de 64 bits, também com sinal.

Instrução	Descrição
mult rs, rt	Realiza a multiplicação entre rs e rt, armazenando internamente (em dois registradores próprios do multiplicador) o resultado.
mfhi rd	Copia a parte alta do resultado da multiplicação (32 bits mais significativos) para o registrador rd.
mflo rd	Copia a parte baixa do resultado da multiplicação (32 bits menos significativos) para o registrador rd.

Importante: Note que o resultado da multiplicação inteira é armazenado em dois registradores. O registrador Hi deve conter os 32 bits mais significativos e o registrador Lo os bits menos significativos.

Função	Operação	Descrição	Flags
000	S = A	Carrega A	Z, N
001	S = A + B	Soma	Z, N, O
010	S = A – B	Subtração	Z, N, O
011	S = A and B	And lógico	Z
100	S = A + 1	Incremento de A	Z, N, O
101	S = not A	Negação de A	Z
110	S = A xor B	OU exclusivo	Z
111	S = A comp B	Comparação	EG, GT, LT



A Unidade Lógica e Aritmética (ALU) é um circuito combinacional que permite a operação com números de 32 bits na notação complemento a dois. A funcionalidade é especificada

pela entrada F conforme descrito na tabela abaixo.

shift	Descrição
000	Nada a fazer.
001	Load no registrador.
010	Shift a esquerda **n** vezes.
011	Shift a direita lógico **n** vezes.
100	Shift a direita aritimético **n** vezes.
101	Rotação a direita **n** vezes.
110	Rotação a esquerda **n** vezes.

O registrador de deslocamento deve ser capaz de deslocar um número inteiro de 32 bits para esquerda e para a direita. No deslocamento a direita o sinal pode ser preservado ou não. O número de deslocamentos pode variar entre 0 e 31 e é especificado na entrada n (5 bits) do registrador de deslocamento. A funcionalidade desejada do registrador de deslocamento é especificada na entrada shift (3 bits menos significativos) conforme figura abaixo.

2.9 - Tratamento de Exceções

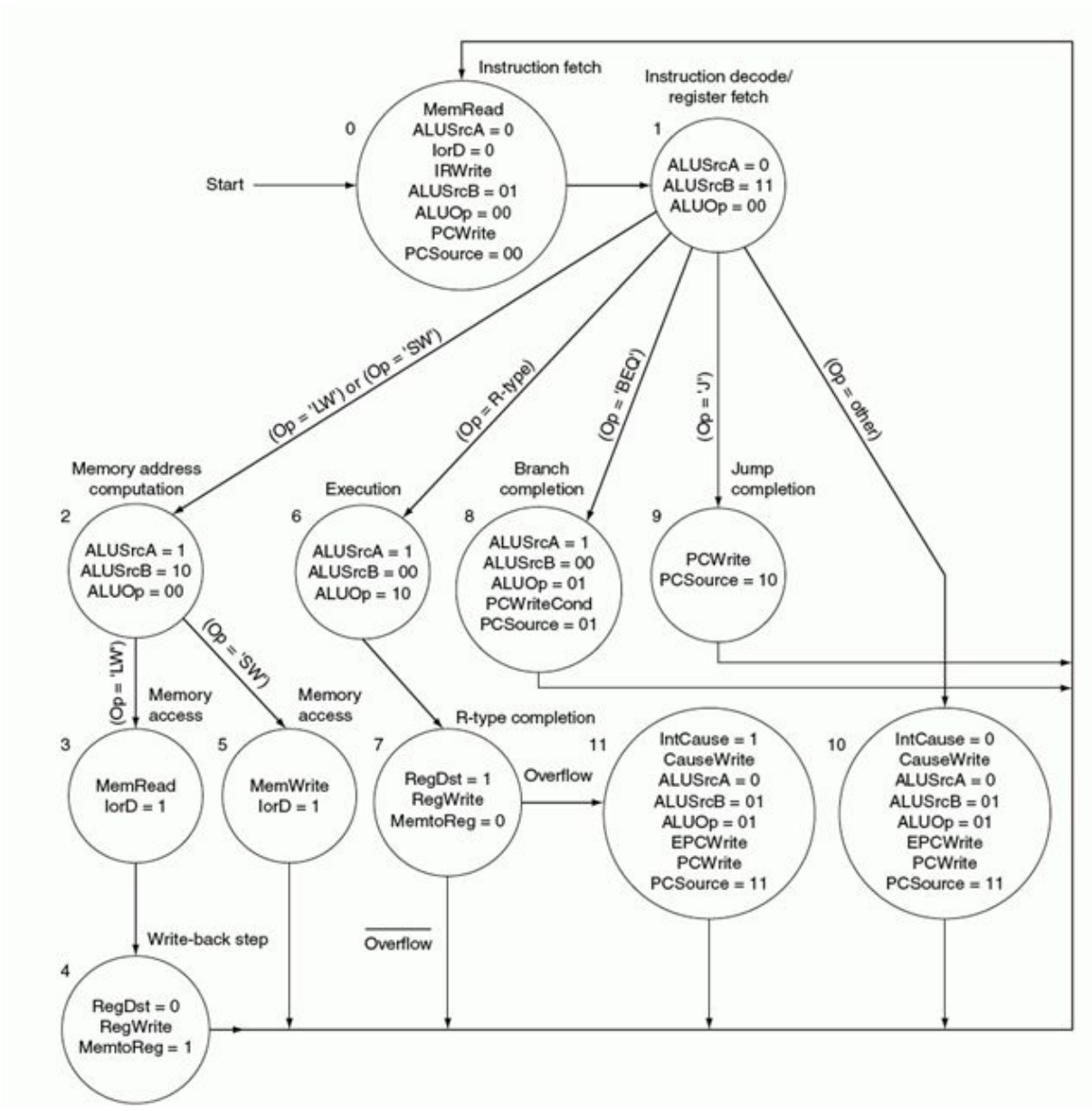
O processador deve incluir tratamento de três tipos de exceções: *opcode inexistente*, *overflow* e *divisão por zero*. Na ocorrência de uma exceção, o endereço da instrução que causou a exceção deverá ser salvo no registrador EPC a ser inserido na unidade de processamento e o PC será carregado, posteriormente, com o valor do endereço da rotina de tratamento. O byte menos significativo do endereço da rotina de tratamento está armazenado nos seguintes endereços de memória: 253 (opcode inexistente) e 254 (overflow) e 255(divisão por zero).

A rotina de tratamento estará armazenada na memória a partir do endereço 228 (decimal) até o endereço 251 (decimal). Esta rotina armazenará o valor 1 no registrador 30 para a execução de um programa no caso de um opcode inexistente. No caso de um overflow, o valor 2 deverá ser armazenado no registrador 30 e a execução do programa deve continuar. Finalmente no caso de divisão por zero, o valor 3 deverá ser armazenado no registrador 30 e a execução do programa finalizada.

Basicamente os passos que devem ser implementados no projeto após ocorrer uma exceção são os seguintes:

1. Ao ser detectada uma exceção o endereço da instrução que a causou deve ser salvo no EPC.
2. O byte localizado na posição de memória 253, 254 ou 255 (dependendo da exceção) deverá ser lido e estendido para 32 bits. Essa extensão é feita apenas com zeros.
3. O número de 32 bits resultante do passo anterior deverá ser o novo valor de PC.
4. Após completar tais passos a rotina de tratamento de exceção que está armazenada na memória será executada, fazendo assim por software a tarefa de armazenar o valor adequado no registrador 30 (O passo 4 não deve ser implementado pela equipe do projeto).

Observação: veja que os passos que acabam de ser descritos não são referentes à instrução rte, pois esta instrução deve apenas armazenar em PC o valor contido atualmente em EPC.


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op = other)
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Overflow
11
IntCause = 1
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
10
IntCause = 0
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
Write-back step
Overflow
4
RegDst = 0
RegWrite
MemtoReg = 1