ARQUITETURA VIKING

Arquitetura feita em 16 bits, estilo Von Neumann

Segue a filosofia RISC

Todos os Espaços no programa são TAB

➔ Registradores

Registrador	Instrução	Apelido	Papel
0	r0	at	Especial
1	r1	r1	Uso Geral
2	r2	r2	Uso Geral
3	r3	r3	Uso Geral
4	r4	r4	Uso Geral
5	r5	sr	Uso Geral
6	r6	lr	Uso Geral
7	r7	sp	Especial

O processador Viking é definido como uma arquitetura baseada em operações de carga e armazenamento(load/store) para acessar à memória de dados.

Para operações Lógicas e aritméticas possam ser executadas, é necessário que os operandos sejam trazidos da memória ou carregados como constantes.

Registradores de Propósito geral(GPRs) : r1 á r6

Registrador Temporário: r0 (at)
Registrador de Ponteiro de Pilha: r7 (sp)

Temporário é usado para é usado para pseudo Operações
Ponteiro de Pilha é usado para armazenamento de dados e chamadas de funções, usado também para desvios incondicionais já que nunca será zero.

Registrador contador de programa(PC): não é mexido pelo programador, serve para apontar a instrução corrente do programa, após uma instrução ser decodificada, PC avança para a próxima instrução

➔ Formatos de Instrução

- Instruções do Tipo R
  Um registrador é definido como destino: Rst
  Dois registradores são definidos como fontes: RsA e RsB

Instrução	Registrador Destino	Registrador Fonte 1	Registrador Fonte 2
add	r3	r1	r2

- Instruções do Tipo I

Um registrador é definido como fonte e destino da Operação: Rst
Segundo valor definido como fonte é obtido do campo Immediate(abreviação: Im), onde é um valor constante.

Instrução	Registrador Fonte e Destino	Imeddiate(será lido em 8 bits)
add	r3	5

Existem 3 modos de endereçamentos no Viking:

- Registrador
  Utiliza somente instruções do tipo R
- Imediato
  Utiliza somente instruções do tipo I
- Relativo ao PC
  Utilizado por instruções do tipo I, classe desvios condicionais.

➔ Conjuntos de Instruções Básicos

- Operadores Lógicos

Instrução	Comando Tipo R	Comando Tipo I	Exemplo R	Exemplo I
AND	and Rst,RsA,RsB	and Rst,Immediate	and r3,r1,r2	and r3,5
OR	or Rst,RsA,RsB	or Rst,Immediate	or r3,r1,r2	or r3,5
XOR	xor Rst,RsA,RsB	xor Rst,Immediate	xor r3,r1,r2	xor,r3,5

- **o** Operadores Sets

Instrução	Explicação	Comando Tipo R	Comando Tipo I
SLT (**set if less than**)	compara dois valores com sinal, se o primeiro for menor que o segundo, armazena 1(true), caso contrário armazena zero(false)	slt Rst,RsA,RsB	slt Rst,Im
SLTU	compara dois valores sem	sltu Rst,RsA,RsB	sltu Rst,Im

(set if less than unsigned)	sinais, se o primeiro for menor que o segundo, armazena 1(true), caso contrário armazena zero(false)		

- **o** Operadores de Adição e Subtração

Instrução	Descrição	Comandos Tipo R	Comandos Tipo I	Exemplo
ADD	Soma os valores dos dois Registradores Origem e salva no Registrador Destino	add Rst,RsA,RsB	add Rst,Im	R: add r3,r1,r2 I: add r3,1
SUB	Subtrai os valores dos dois Registradores e salva no Registrador Destino	sub Rst,RsA,RsB	sub Rst,Im	R: sub r3,r1,r2 I: sub r3,1

Instrução	Explicação	Comando
ADC (add with Carry)	Soma dois valores e armazena o resultado em um registrador e também traz o Carry gerado pela ultima instrução	adc Rst,RsA,RsB
SBC (sub with Carry)	Subtrai dois valores e armazena o resultado em um registrador e também trás o Carry gerado pela última instrução	sbc Rst,RsA,RsB

- **o** Operações de Carregamento(Load)
  - Load do Tipo I apenas

Instrução	Descrição	Comando
LDR (Load Register)	Carrega um valor para um registrador com sinal (± 128)	ldr Rst,Im
LDC (Load Constant)	Carrega um valor para um registrador sem sinal (aceita maiores que ± 128)	ldc Rst,Im

- Load do Tipo R apenas

Instrução	Descrição	Comando
LDB (Load Byte)	Carrega um byte da memória, o endereço é obtido do RsB, carregado no Rst e o sinal fica no r0(at).	ldb Rst,at,RsB

LDW (Load Word)	Carrega uma Palavra da Memória,podendo ser um valor vindo de uma variavel	ldw Rst,RsB
		ou
		ldw Rst,variavel

- Operadores de Carga(Store)

Instrução	Descrição	Comando
STB (Store Byte)	Armazena um Byte na memória, o endereço vem do RsA e vai para o RsB	stb RsA,RsB
STW (Store Word)	Armazena uma palavra na memória, o endereço é obtido a partir do RsB, o valor armazenado se encontra no RsA	stw RsA,RsB

- Operadores de Desvios Condicionais

Instrução	Descrição	Comando Tipo R	Comando Tipo I
BEZ (branch if equal zero)	Caso o valor de RsA seja zero, ele irá pegar o valor de RsB	bez RsA,RsB	bez Rst,Im
BNZ (branch if not equal zero)	Caso o valor de RsA seja diferente de zero, ele irá pegar o valor de RsB	bnz RsA,RsB	bnz Rst,Im

- Operadores de Deslocamento e Rotação

Instrução	Descrição	Comando
LSR (Logical Shift Right)	Realiza o deslocamento lógico por 1 bit a direita e armazena em um Registrador, possui um carry tbm	lsr Rst,RsA,r0
ASR (Arithmetic shift Right)	Realiza o deslocamento aritmético por 1 bit á direita e armazena em um registrador	asr Rst,RsA,r0
ROR (Rotate right	Realiza a rotação por 1 bit á	ror Rst,RsA,r0

through carry)	direita e armazena em um registrador, o valor inserido no bit mais significativo é o valor de carry gerado na ultima instrução	

➔ Pseudo Operações

Existem instruções que não fazem parte da arquitetura viking, mas são instruções tipicamente encontradas na arquitetura RISC e servem para facilitar no desenvolvimento de programas em linguagem de montagem(Assembly), sendo uma instrução que simplifica uma camada de instruções básicas

- **o** Glossário

at	Registrador temporário(r0)
lr	Endereço de Retorno
const	Parâmetro para um valor numérico ou uma variável(rótulo)
addr	Parâmetro que pode ser uma variável(rótulo)
r1	é o registrador Destino(Rst)
r2	é o registrador Fonte(RsA)

- Lista de Comandos de Pseudo Operações

Instrução	Descrição	Formato
NOP	No Operation	nop
NOT	One´s Complement	not r1
NEG	Two´s Complement	neg r2
MOV	Move Register	mov r1,r2
LSR	Logical Shift right	lsr r1,r2
ASR	Arithmetic shift right	asr r1,r2
ROR	Rotate right through carry	ror r1,r2
LSL	Logical shift left	lsl r1,r2
ROL	Rotate left through carry	rol r1,r2
LDI	Load immediate	ldi r1,const
BEZ	Branch if equal zero	bez r1,r2 bez r1,addr
BNZ	Branch if not qual zero	bnz r1,r2 bnz r1,addr
LDB	Load Byte	ldb r1,r2 ldb r1,addr
STB	Store Byte	stb r1,r2 stb r1,addr
LDW	Load Word	ldw r1,r2 ldw r1,addr
STW	Store Word	stw r1,r2 stw r1,addr
HCF	Halt and Catch fire	hcf

➔ Programando com o Processador Viking

- SELEÇÃO (if...else)

  - **Igual a (==)**

    Se os dois valores nos registradores r1 e r2 forem iguais, o programa irá ativar a condição if

```
main
     ldw  r1,a
     ldw  r2,b
     sub  r3,r1,r2
     bez  r3,if
else
     ...
if
     ...

a    2
b    2
```

  - **Diferente de (!=)**

    Se os dois valores nos registradores r1 e r2 forem diferentes, o programa irá ativar a condição if

```
main
   ldw  r1,a
   ldw  r2,b
   sub  r3,r1,r2
   bnz  r3,if
else
   ...
if
   ...

a  2
b  3
```

  - **Menor que (<)**

    Se os valores forem sinalizados(negação), usa-se SLT, se os valores não forem sinalizados, usa-se o SLTU.

    Utiliza-se o comando BNZ, para comparar os valores nos registradores e se o valor de r1 for menor que r2, o resultado da comparação será diferente de zero e o if será ativado

```
main
```

```
     ldw  r1,a
     ldw  r2,b
     slt  r3,r1,r2
     bnz  r3,if
else
     ...
if
     ...

a    2
b    4
```

- **Maior que (>)**

essa instrução segue o modelo do menor que, onde colocamos a chamada do registrador r2 antes do r1

```
main
   ldw  r1,a
   ldw  r2,b
   slt  r3,r2,r1
   bnz  r3,if
else
  ...
if
  ...

a  2
b  4
```

- **Maior ou Igual a (>=)**

Ele segue a mesma lógica do menor que, a única diferença é que se ele não é menor que um numero, ele só pode ser maior ou igual a outro, então usamos a instrução BEZ para iniciar o if

```
main
```

```
  ldw   r1,a
  ldw   r2,b
  slt   r3,r1,r2
  bez   r3,if
else
  ...
if
  ...

a 4
b 4
```

- **Menor ou Igual a (<=)**

Ele segue a mesa lógica do Maior ou igual a, mas a única diferença é o registrador que será pego primeiro, sendo nesse caso o registrador r2

```
main
  ldw   r1,a
  ldw   r2,b
  slt   r3,r2,r1
  bez   r3,if
else
  ...
if
  ...


a 4
b 4
```

- REPETIÇÃO(while)

- **Repetição incondicional**

é uma repetição em loop eterno

while(1){...}

```
main
while
```

```
    ...
    bnz r7,while
endwhile
    hcf
```

- **Repetição Condicional**

  É quando você diz uma condição de parada para a repetição

  Exemplo: enquanto a != 0 ele irá permanecer no while, se ele for zero ele conclui o while e termina

```
main
   ldw  r1,a
while
   sub  r1,1
   bez  r1,endwhile
   bnz  r7,while
endwhile
   hcf
a  3
```

- VARIÁVEIS

  As variáveis são criadas no final do código, mas devem ser chamadas da memória para os registradores disponíveis desejados usando o comando LDW e se quiser armazenar um valor em uma variável utiliza-se o comando STW.

  Exemplo: somar dois valores e salvar em uma terceira variável:

```
; isto é um comentário

main
       ldw  r1,a ;carrega o valor da variável a
```

```
        ldw  r2,b ;carrega o valor da variável b
        add  r3,r1,r2 ;faz a soma dos dois valores
                      ;e armazena em r3
        stw  r3,c ;guarda o valor do registrador r3
                  ;dentro da variável c
        hcf ;para o programa

a 2
b 3
c 0 ;a variável c foi iniciada com zero
```

- **Imprimindo valores no Terminal**

  Para imprimir valores no terminal, usaremos um comando pré-definido na arquitetura de 16 bits, chamado Mapa de Memória, como os abaixo:

Papel	código(16 bits)
Código + Dados(inicio)	0x0000
Ponteiro de Pilha	0xdffe
Saída(caracter)	0xf000
Saída(inteiro)	0xf002
Entrada(caracter)	0xf004
Entrada(inteiro)	0xf006

  - imprimindo valores no terminal:
    exemplo: saída da soma de dois valores

```
    main
        ldw  r1,a
        ldw  r2,b
        add  r3,r1,r2
        ldi  at,0xf002 ;irá carregar no r0 o command
        stw  r3,at ;armazena o valor de r3 no r0
        hcf

    a  2
    b  3
```