

Implementação do Processador RS5



Fernando Gehm Moraes - fernando.moraes@pucrs.br

Willian Analdo Nunes - willian.nunes@edu.pucrs.br

Angelo Dal Zotto - angelo.dalzotto@edu.pucrs.br



Risc-V



Origem

- Desenvolvido na Universidade da Califórnia, Berkeley, em 2010

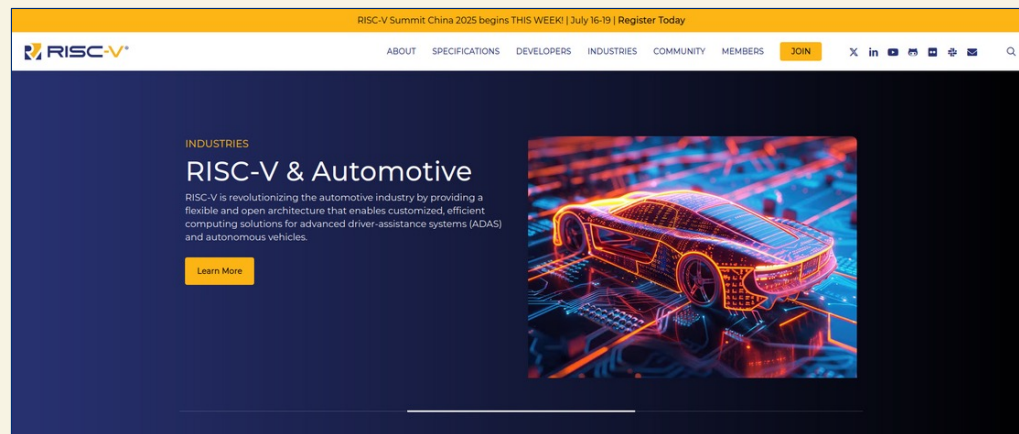
Motivação

- Criar uma arquitetura (ISA) aberta e livre de royalties
- Solucionar problemas de licenciamento e limitações das ISAs proprietárias
- Prover uma base modular e extensível para pesquisa e implementação comercial

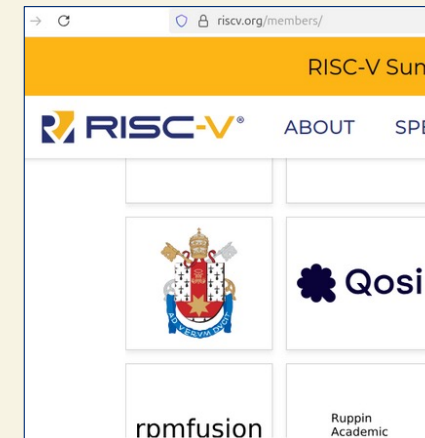
Manutenção atual do RISC-V

- Fundação RISC-V International
 - Organização sem fins lucrativos
 - Criada para padronizar e promover a ISA RISC-V
- Membros
 - Empresas, universidades e organizações globais
 - Inclui líderes da indústria como Intel, NVIDIA, Google e Alibaba

<https://riscv.org/>



<https://lists.riscv.org/g/community-org-members>



About RISC-V International

RISC-V International is the global non-profit home of the open standard RISC-V Instruction Set Architecture (ISA), related specifications, and stakeholder community.

JOIN RISC-V INTERNATIONAL

GET IN TOUCH

Ratified Specification

The RISC-V open-standard instruction set architecture (ISA) defines the fundamental guidelines for designing and implementing RISC-V processors.

VIEW RATIFIED SPECS

SPECS UNDER DEVELOPMENT

The RISC-V ISA specifications, extensions, and supporting documents are collaboratively developed, ratified, and maintained by contributing members of RISC-V International.

These specifications are all free and publicly available.

[View Original Specifications »](#)

Available Specifications

- [Ratified ISA Specifications](#)
- [Ratified Profiles Specification](#)
- [Ratified Non-ISA Specifications](#)
- [Ratified Extensions](#)
- [Additional Documents](#)
- [Under Development](#)

Extensões e Modularidade



ISA Base

- **RV32I** e **RV64I**: conjuntos de instruções mínimos para arquitetura de 32 e 64 bits
- Propositadamente simples para facilitar implementações

Extensões Padronizadas

- **M**: multiplicação e divisão de inteiros
- **A**: instruções atômicas
- **F/D**: ponto flutuante de precisão simples e dupla
- **C**: compactação de instruções para eficiência de espaço
- **G** (General-Purpose): **I + M + A + F/D** (exemplos: RV32G e RV64G)

Extensões Personalizadas

- flexibilidade para criar extensões específicas sem comprometer a compatibilidade com o padrão

Arquitetura Privilegiada do RISC-V



- Define os modos de operação e os mecanismos necessários para gerenciar o hardware, executar o sistema operacional e controlar o acesso aos recursos do sistema
- Esses modos são fundamentais para a segurança e o gerenciamento eficiente do processador

Modos de Privilegiados

1. Machine Mode (M)

- a. O nível mais alto de privilégio
- b. Tem controle total do hardware
- c. Normalmente usado para inicialização e configuração do sistema

2. Supervisor Mode (S)

- a. Usado por sistemas operacionais para gerenciar recursos
- b. Controla operações como gerenciamento de memória

3. User Mode (U):

- a. O nível mais baixo de privilégio
- b. Executa aplicativos sem acesso direto a recursos críticos do sistema
- c. Depende do modo Supervisor ou Machine para acesso seguro ao hardware

Level	Encoding	Name	Abbreviation
0	00	User/Application	U
1	01	Supervisor	S
2	10	<i>Reserved</i>	
3	11	Machine	M

Importância da arquitetura privilegiada

- **Segurança:** impede que programas em modo U interfiram diretamente no sistema operacional ou em outros processos
- **Flexibilidade:** facilita o desenvolvimento de sistemas operacionais

Arquitetura Privilegiada do RISC-V



Control and Status Registers (CSRs)

- registradores especializados usados para armazenar e controlar informações relacionadas ao estado do sistema e gerenciamento de exceções/interrupções

- Exemplos importantes incluem:

1. **mstatus** (Machine Status)

- controla habilitações de interrupção e estados de privilégio
- bits importantes: MIE (habilita interrupções no modo M), MPP (armazena o privilégio anterior ao trap)

2. **mtvec** (Machine Trap-Vector Base Address)

- define o endereço do manipulador de traps no modo máquina

3. **mepc** (Machine Exception Program Counter)

- armazena o endereço da próxima instrução após a ocorrência de uma trap

4. **mcause**

- indica a causa de uma interrupção ou exceção

5. **mscratch**

- usado como espaço temporário para salvamento de contexto

Arquitetura Privilegiada do RISC-V

Manejo de Traps (Interrupções e Exceções)



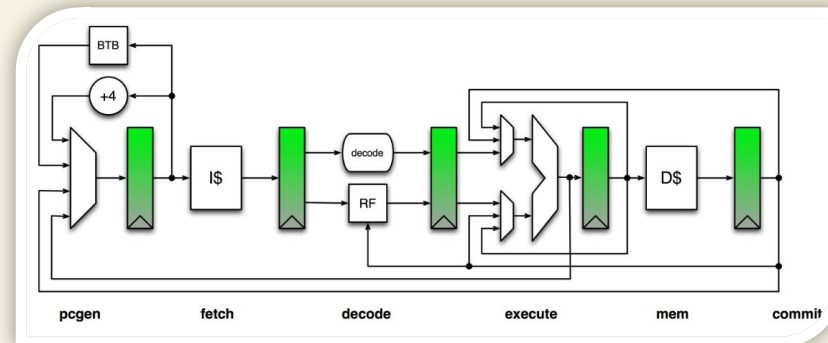
- Trap Handling: envolve a captura de eventos que desviam o fluxo normal de execução, como exceções ou interrupções
- **Os principais passos incluem:**
 1. **Ocorrência do Trap:**
 - Pode ser uma interrupção externa (ex.: timer ou GPIO) ou uma exceção (ex.: acesso ilegal à memória)
 2. **Salvamento de Contexto:**
 - O PC atual é salvo em **mepc**
 - O estado de execução (privilegio atual, interrupções habilitadas) é salvo em **mstatus**
 3. **Alteração de Privilégio:**
 - O sistema troca para o nível apropriado (ex.: U para M) usando o campo **MPP**
 - O trap handler é executado a partir do endereço definido em **mtvec**
 4. **Execução do Trap Handler:**
 - O código no manipulador pode consultar **mcause** para determinar o motivo do trap e tomar as ações apropriadas
 5. **Restauração de Contexto e Retorno:**
 - Após tratar o evento, os registradores salvos são restaurados
 - O controle é devolvido ao código original com a instrução mret (Machine Return)

Ver exemplo em: <https://github.com/gaph-pucrs/MAestro/blob/76700d2068dfe205be5766dcea890cb45df9c5a1/hal/hal.S>

Processadores RISC-V

Exemplos de distribuição **open-source**:

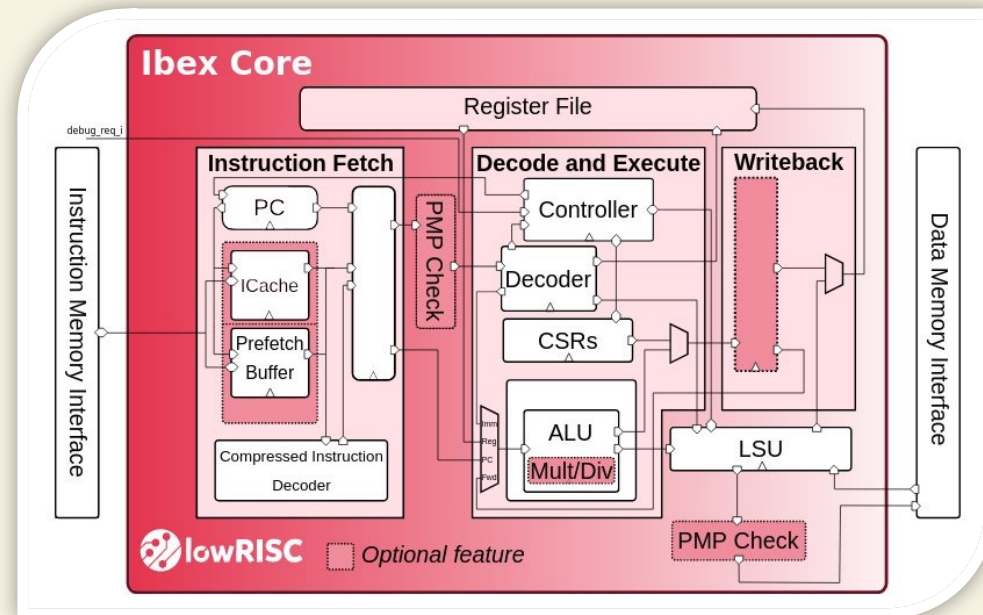
- ❑ Rocket
 - 5 estágios
 - RV64G



Processadores RISC-V

Exemplos de distribuição **open-source**:

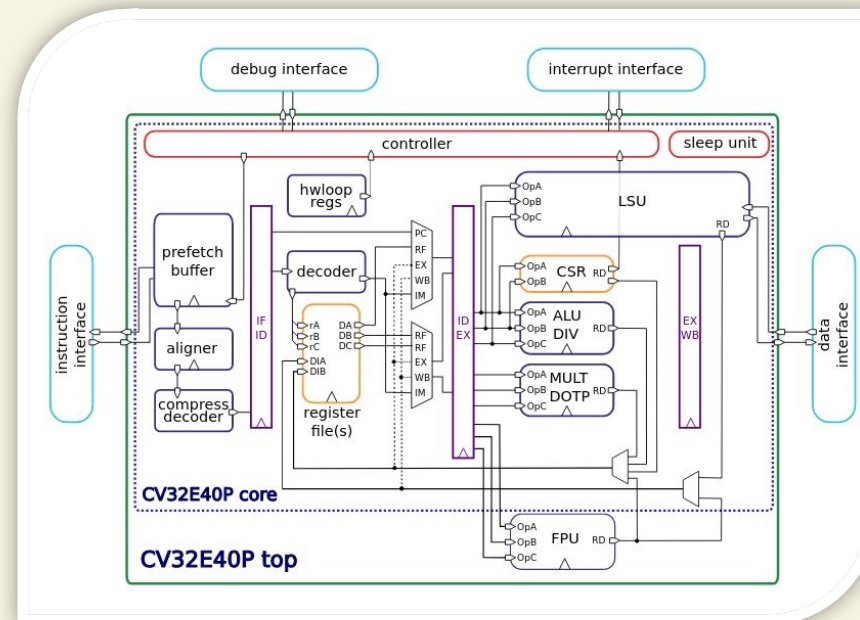
- ❑ Rocket
 - 5 estágios
 - RV64G
- ❑ Ibex
 - 2 Estágios
 - RV32(I/E)MCB



Processadores RISC-V

Exemplos de distribuição **open-source**:

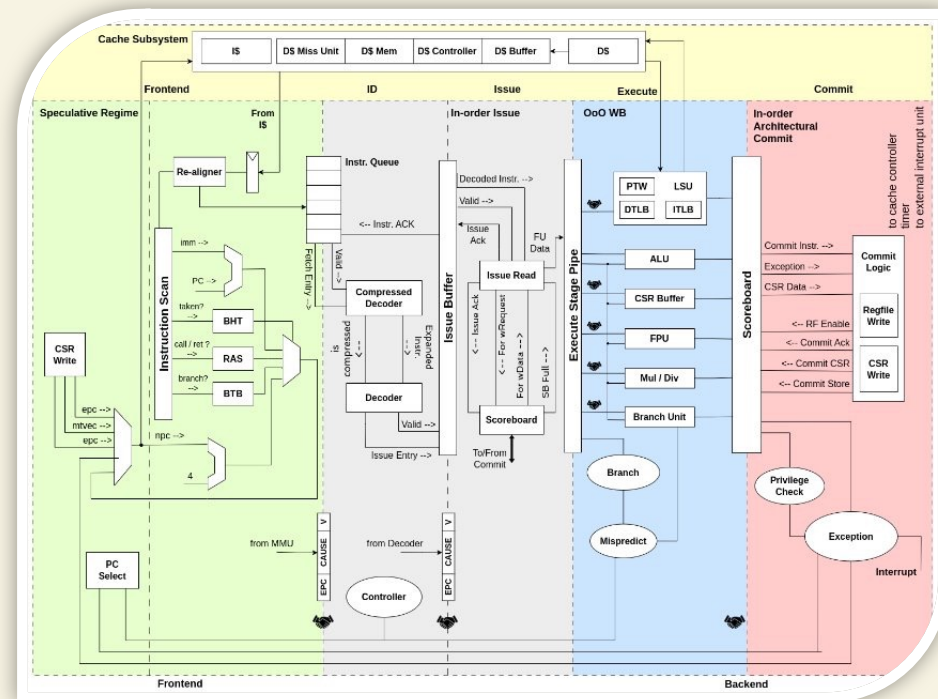
- ❑ Rocket
 - 5 estágios
 - RV64G
- ❑ Ibex
 - 2 Estágios
 - RV32(I/E)MCB
- ❑ CV32E40P
 - 4 Estágios
 - RV32IMC(F_Zfinx) + Custom



Processadores RISC-V

Exemplos de distribuição **open-source**:

- ❑ Rocket
 - 5 estágios
 - RV64G
- ❑ Ibex
 - 2 Estágios
 - RV32(I/E)MCB
- ❑ CV32E40P
 - 4 Estágios
 - RV32IMC(F_Zfinx) + Custom
- ❑ CVA6 (ARIANE)
 - 6 Estágios
 - RV64IMAC



Processadores RISC-V

Exemplos de distribuição **open-source**:

❑ Rocket

- 5 estágios
- RV64G

❑ Ibex

- 2 Estágios
- RV32(I/E)MCB

❑ CV32E40P

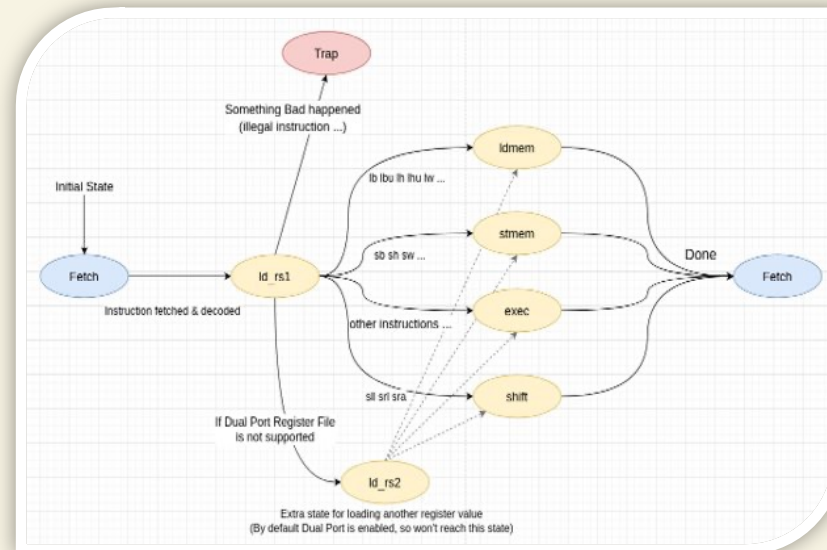
- 4 Estágios
- RV32IMC(F_Zfinx) + Custom

❑ CVA6 (ARIANE)

- 6 Estágios
- RV64IMAC

❑ PICORV32

- Multiciclo
- RV32IMC



Processadores RISC-V - Indústria

- SiFive → <https://www.sifive.com>
- Alibaba → <https://riscv.org/ecosystem-news/2021/10/alibaba-announces-open-source-risc-v-based-xuantie-series-processors-pandaily/>
- Esperanto Technologies → <https://www.esperanto.ai>
 - ET-SoC-1 Chip: thousand RISC-V processors on a single TSMC 7nm chip
- NVIDIA → https://riscv.org/wp-content/uploads/2024/12/Tue1100_Nvidia_RISCV_Story_V2.pdf
- Microchip
 - Multi-Core RISC-V SoC FPGAs -> <https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas/polarfire-soc-fpgas>

Processador RS5 (PUCRS)

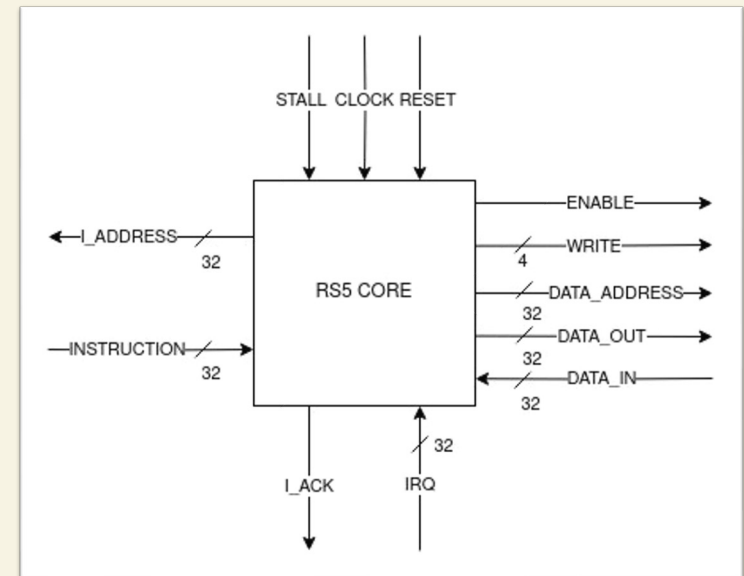
- RS5 é um processador que implementa o [RV32I](#), com pipeline de 4 estágios, com diversas extensões, e suporte a modo Máquina e Usuário
- Descrito em SystemVerilog
- Suporte a sistemas operacionais embarcados (e.g. Zephyr)
- Integra componentes de hardware e software

Interface com a memória:

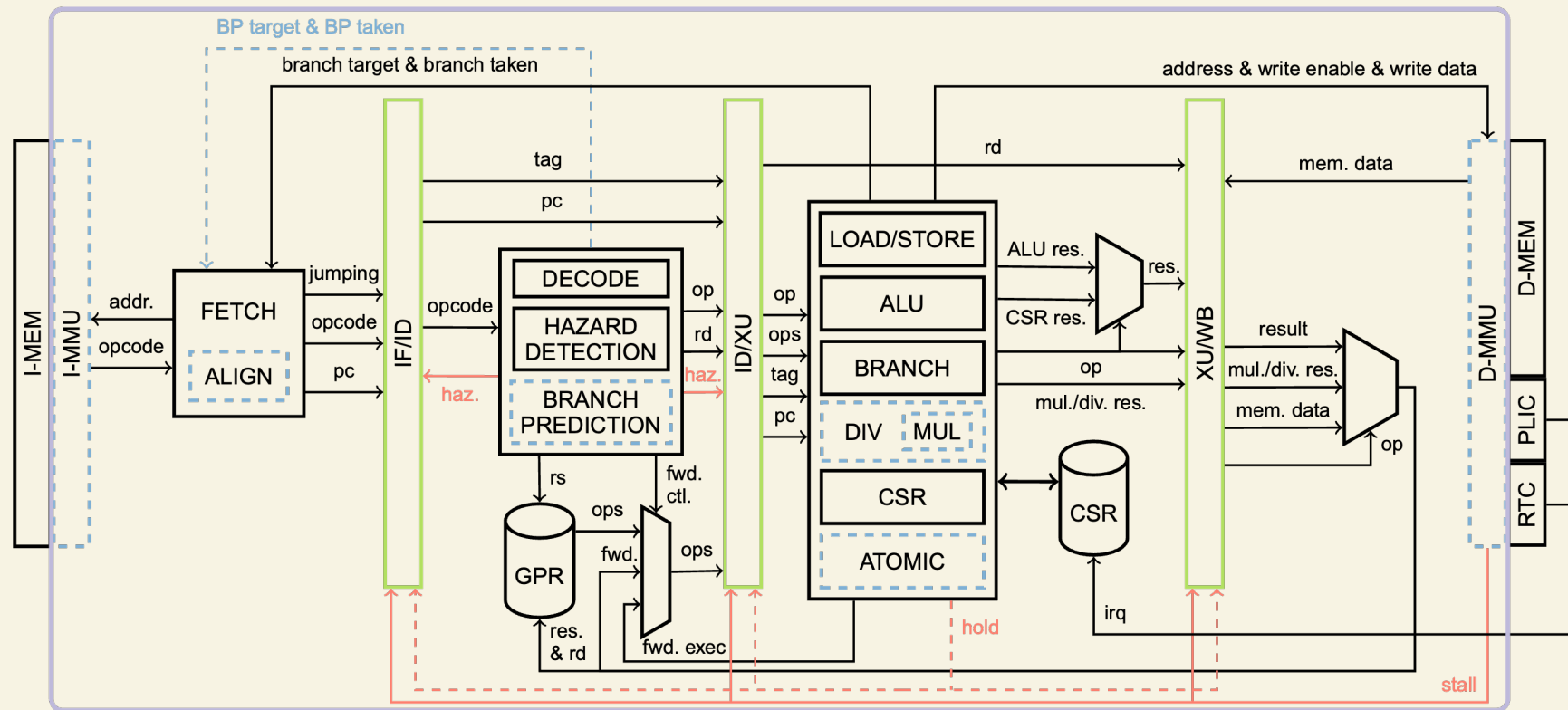
- Dual Port Memory
- Uma porta para Instruções
- Uma porta para Dados

Principais desenvolvedores

- Willian Nunes
- Angelo Dal Zotto



RS5 - Visão geral



RS5 - parâmetros de projeto

RS5 oferece parâmetros de projeto para definir a inclusão de recursos opcionais (extensões e funcionalidades)

- **Environment:** permite alternar rapidamente entre construções ASIC e FPGA
- **MULEXT:** seleciona a inclusão de aceleração para multiplicação e divisão (Extensão M) ou apenas multiplicação (Extensão Zmmul)
- **AMOEXT:** extensão atômica que adiciona instruções para operações de leitura-modificação-gravação de memória no mesmo ciclo
- **COMPRESSED:** extensão comprimida que suporta instruções de 16 bits, reduzindo o uso de memória
- **XOSVMEEnable:** unidade de Gerenciamento de Memória (MMU) personalizada na forma de uma extensão chamada XOSVM (Offset and Limit Virtual Memory)
- **ZIHPMEnable:** configuração opcional que inclui monitores de desempenho para profiling de instruções
- **ZKNEEnable:** extensão opcional que acelera o algoritmo de criptografia **AES**
- **BRANCHPRED:** preditor de desvio que melhora o desempenho
- **VENABLE:** extensão opcional que adiciona capacidades de **processamento vetorial**
- **VLEN:** define o tamanho dos registradores na unidade vetorial quando a extensão vetorial está habilitada

RS5 - parâmetros de projeto

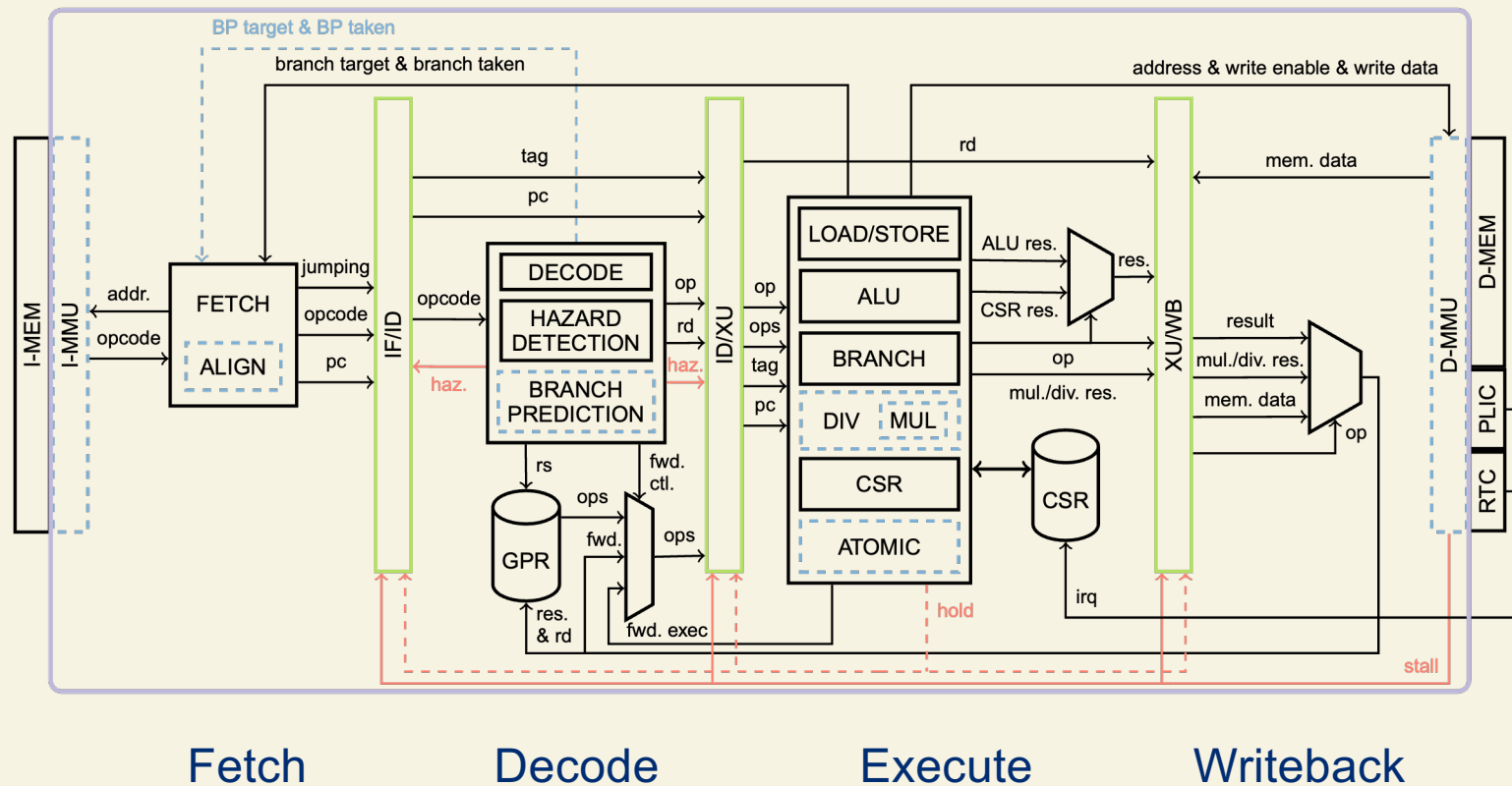
Parameter	Description	Options
Environment	Environment type	ASIC, FPGA
MULEXT	Include Hardware Multiplication/Division extension	MUL_OFF, MUL_ZMMUL, MUL_M
AMOEXT	Include Atomic operation extension	AMO_OFF, AMO_ZALRSC, AMO_ZAAMO, AMO_A
COMPRESSED	Include Compressed extension	TRUE, FALSE
XOSVMEEnable	Include XOSVM extension (MMU)	TRUE, FALSE
ZIHPMEnable	Include ZIHPM extension (Performance Monitors)	TRUE, FALSE
ZKNEEnable	Include ZKNE extension (AES Hardware acceleration)	TRUE, FALSE
BRANCHPRED	Include Branch prediction	TRUE, FALSE
VEnable	Include Vector extension	TRUE, FALSE
VLEN	Vector length in bits	64, 128, 256, ...

RS5 - top default

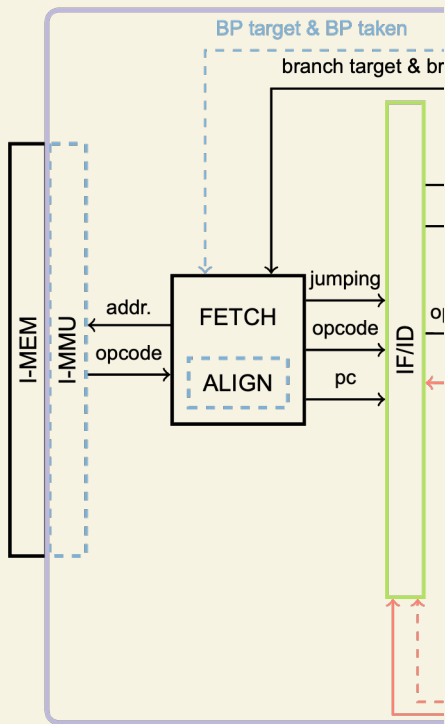
```
module RS5
|   import RS5_pkg::*;
#(
`ifndef SYNTH
|   parameter bit          DEBUG          = 1'b0,
|   parameter string       DBG_REG_FILE   = "./debug/regBank.txt",
|   parameter bit          PROFILING      = 1'b0,
|   parameter string       PROFILING_FILE = "./debug/Report.txt",
`endif
|   parameter environment_e Environment = ASIC,
|   parameter mul_e        MULEXT       = MUL_M,
|   parameter atomic_e     AMOEXT       = AMO_A,
|   parameter bit          COMPRESSED    = 1'b0,
|   parameter bit          VEnable       = 1'b0,
|   parameter int          VLEN          = 256,
|   parameter bit          XOSVMEEnable  = 1'b0,
|   parameter bit          ZIHPMEnable   = 1'b0,
|   parameter bit          ZKNEEnable    = 1'b0,
|   parameter bit          BRANCHPRED    = 1'b1
)

```

Pipeline - 4 estágios



Pipeline - Fetch (IF)



Fetch

Controla a recepção de instruções da memória de instruções (I-MEM)

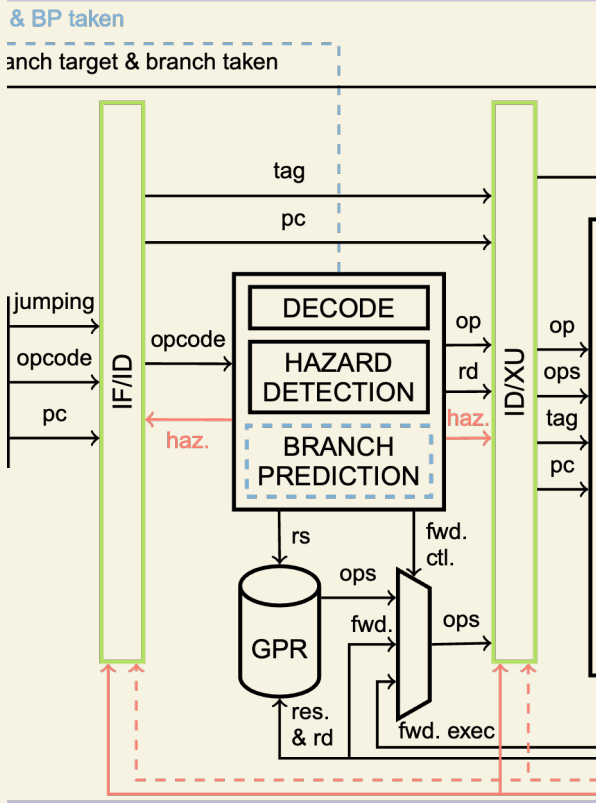
Selecciona o **Program Counter (PC)** com base nos seguintes eventos:

- Reset
- Ocorrência de traps
- Retorno de traps
- Jumps
- Operação sequencial (default)

Pode operar com endereços de memória virtual, se conectado a uma I-MMU para traduzir endereços virtuais para físicos (extensão XOSVM)

Com Extensão Comprimida: inclui um alinhador de endereços. Contém um descompressor de instruções que transforma instruções de 16 bits em 32 bits para decodificação

Pipeline - Decode (ID)



Decode

- Extrai a operação (**op**) em função da instrução lida no fetch
 - Identifica o registrador de destino (**rd**)
 - Decodifica os registradores de origem (**rs**)
 - Busca os operandos (**ops**) no banco de registradores (**GPR**)
 - Realiza a detecção de hazards
-
- **Forwarding**
 - Identifica se um **rs** da instrução atual é igual ao **rd** está sendo escrito no estágio writeback ou execute, evitando esperar pela gravação do banco de registradores, ignorando assim os dados ops
-
- **Detecção de Hazard de Dados**
 - Emite operações nulas (**haz.**), inserindo bolhas no pipeline, aguardando os ciclos necessários para resolver o conflito
-
- **Branch Predictor** (opcional)
 - Gera sinais conectados diretamente ao primeiro estágio: **BP target** e **BP taken**.

Pipeline - Execute (XU) 1/2

Responsável pela execução da instrução, usando *op* e *ops*.

Unidades de Execução:

1. **ALU:**

- gera **ALU res**
- operações lógicas e aritméticas, endereço de acesso à memória (dados) e endereço de saltos

2. **BRANCH:** gerencia desvios condicionais

3. **LOAD/STORE:** gerencia acessos à memória de dados (D-MEM)

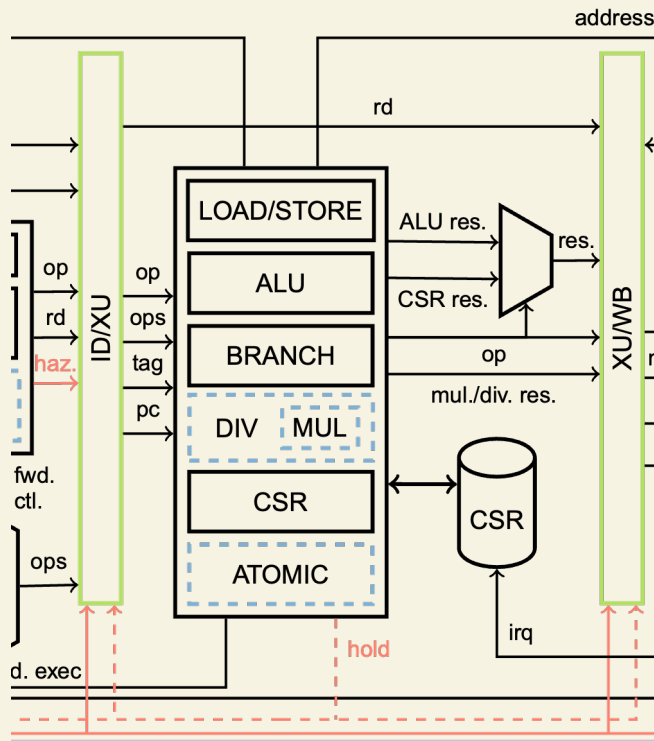
4. **CSR:**

- executa operações **atômicas** em CSR
- gerencia privilégios e traps

5. **Multiplicação (MUL) e Divisão (DIV)** – opcional (extensão Zmmul):

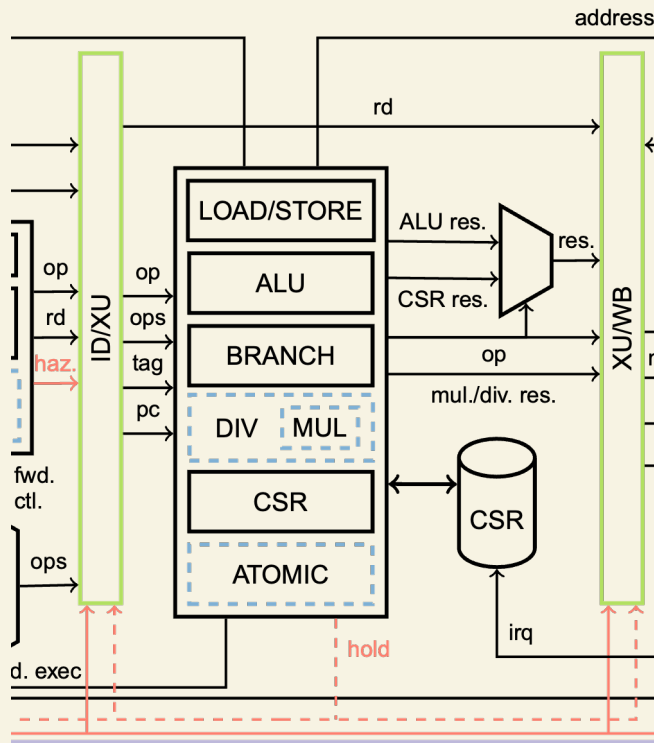
- Inclusão da unidade de divisão implica inclusão da multiplicação (extensão M)

6. **ATOMIC** (opcional): controla operações de leitura-modificação-gravação na memória



Execute

Pipeline - Execute (XU) 2/2



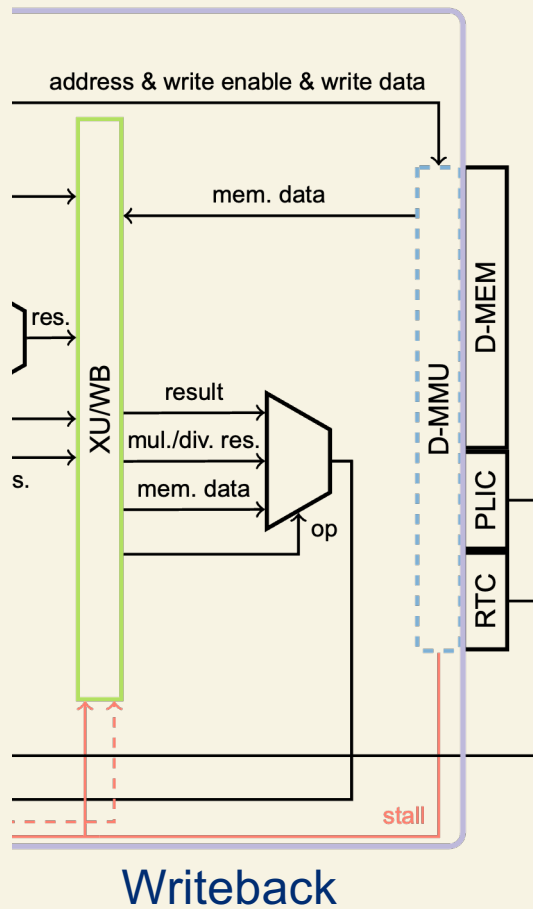
Execute

Escrita na Memória: realizada no estágio XU

Gerenciamento de Operações Multi-Ciclo: o estágio XU controla o sinal **hold** para operações de múltiplos ciclos:

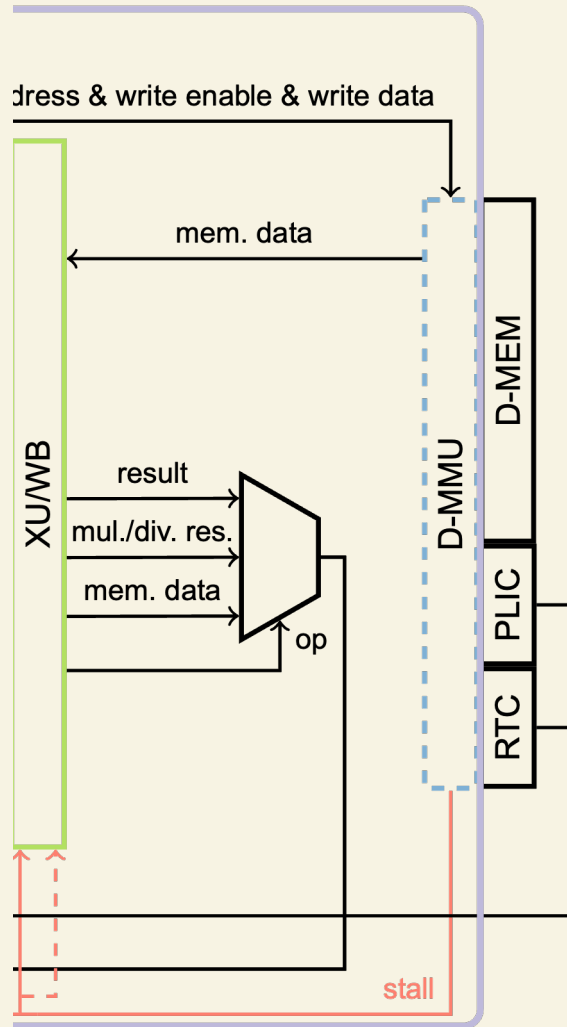
- **Multiplicação:** geralmente requer de 4 a 5 ciclos
- **Divisão:** pode levar até 32 ciclos
- **Operações Atômicas:** também gerenciadas pelo sinal hold

Pipeline - Writeback (WB)



- Executa a escrita dos resultados de **op**
- Escreve os dados obtidos da memória de dados (**D-MEM**) após leitura solicitada pela unidade LOAD/STORE
- Opcional: uma D-MMU pode virtualizar os endereços da D-MEM

PLIC e RTC



PLIC (Platform Level Interrupt Controller)

- Controlador de interrupções

RTC (Real Time Clock)

- Interrupções por timer

Externos ao RS5 – operam com registradores mapeados em memória

RS5 - AVALIAÇÃO (LASCAS 24)

RS5 EVALUATION AND COMPARISON WITH SIMILAR CORES
(Z*:ZICNTR, D*: ZIHPM, X*: XOSVM).

Core	LUTs	FFs	DSPs	Extensions	Modes	Freq. MHz	CoreMark
RS5 Baseline	2141	957	-	Z*	M, U	100	86.3
RS5	2380	1466	-	Z*, D*	M, U	100	86.3
	2553	1032	-	Z*, X*	M, U	100	86.3
	2222	842	12	Z*, Zmmul	M, U	100	212.3
	2814	1113	12	Z*, M	M, U	100	212.3
	3574	1757	12	Z*, M, X*, D*	M, U	100	212.3
Ibex	2184	1247	-	Z*, C	M, U	50	46.8
	2688	1329	1	Z*, M, C	M, U	50	111.6
Steel	2140	1434	-	Z*	M	50	68.0
RS5 - no LUTRAM	2721	1949	-	Z*	M, U	100	86.3
	3395	2105	12	Z*, M	M, U	100	212.3
SCR1	2938	1617	-	-	M	66	70.4
	3518	1747	4	M	M	50	114.3
CV32E40P	5111	2015	5	Z*, M, C	M, U	70	186.8

Obtendo e configurando o RS5 (1/2)

Obtendo os fontes do projeto:

Para isto abra seu terminal, crie ou encontre uma pasta e cole o comando:

- git clone <https://github.com/gaph-pucrs/RS5>

Observação:

- Para utilizar o comando git é necessário tê-lo instalado <https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-Instalando-o-Git>

O comando acima faz o clone do repositório do projeto para seu ambiente local.

Após finalizado acesse a pasta RS5, onde você verá a seguinte estrutura:

```
➤ Tree - L1
├── app           ➡ códigos exemplos
├── docs
├── imgs
├── LICENSE
├── proto        ➡ projeto para o Vivado
├── README.md
├── rtl          ➡ RTL do processador
└── sim         ➡ scripts de simulação
```

Obtendo e configurando o RS5 (2/2)

Compilar programas:

Para conseguirmos compilar e utilizar nossos próximos exercícios é necessária a configuração do compilador cruzado

Para isto siga o passo a passo disponível em:

- <https://github.com/gaph-pucrs/Memphis-V/blob/master/docs/riscv.md>

Atenção: no sistema operacional sendo utilizado, o tutorial disponibiliza instalação para:

- RHEL (dnf-based distros)
- Ubuntu (apt-based distros)
- Arch (pacman/aur distros)

Simular Programas:

Para gerar a simulação de nossos códigos será necessário ter instalado o **Verilator**. O código atual especifica este programa.

Executando Simulações:

Para vermos nossa simulação, o formato de ondas do sinais e resultados, se faz necessário a instalação de uma ferramenta como o **QuestaSim**, ou **ModelSim**, caso você esteja habituado a rodar simulações poderá utilizar o programa de sua preferência.

Exemplo 1 - Assembly (1/3)

Exemplo: app/fibonacci-asm/src/fibonacci.S

1	<code>.section .init</code>	→	Seção de inicialização
2	<code>.global _start</code>	→	Exportação de símbolo para o compilador
3	<code>_start:</code>	→	Símbolo padrão de entrada (início - entry point)
4	<code> # Configure system status MPP=0, MPIE=0, MIE=0</code>		
5	<code> csrw mstatus, zero</code>		
6			
7	<code> # Mask interrupts</code>		
8	<code> csrw mie, zero</code>		
9			
10	<code> # Clear pending interrupts</code>		
11	<code> csrw mip, zero</code>		
12			
13	<code> # Set stack to top of data memory</code>	→	Define stack pointer de acordo com MEM_SIZE no Makefile
14	<code> li sp, MEM_SIZE</code>		
15			
16	<code> jal main</code>	→	Salta para main
17	<code>.end:</code>		
18	<code> li t0, 0x80000000</code>		
19	<code> sw zero, 0(t0) # End simulation</code>		
20	<code> j .end</code>		

Desabilita interrupções

Sinalização para o test bench que a simulação terminou.

Exemplo 1 - Assembly (2/3)

Exemplo: app/fibonacci-asm/src/fibonacci.S

22	<code>.section .text</code>	→	Seção de instruções (text)
23	<code>main:</code>		
24	<code>addi sp, sp, -4</code>		
25	<code>sw ra, 0(sp)</code>	→	Empilha endereço de retorno para label .end (ra é modificado pelo jal)
26			
27	<code>li a0, 5</code>	}	a0: argumento 0 (valor 5) para a função fibonacci
28	<code>jal fibonacci</code>		
29			
30	<code>lw ra, 0(sp)</code>	→	Desempilha o ra referente ao .end
31	<code>addi sp, sp, 4</code>		
32			
33	<code>li t0, 0x80002000</code>		
34	<code>sw a0, 0(t0)</code>	→	Grava o retorno (a0) em um endereço mapeado em memória
35			
36	<code>ret</code>	→	Volta para seção de início (label .end)

Exemplo 1 - Assembly (3/3)

Exemplo: `app/fibonacci-asm/src/fibonacci.S`

```
38  fibonacci:
39      li  t1, 1 # (n - 1)
40      ble a0, t1, finish # if (n <= 1) return
41
42      mv  t0, zero # current
43      mv  t2, zero # (n - 2)
44
45      li  t3, 2    # iterator
46  loop:
47      add  t0, t1, t2 # current = fib(n-1) + fib(n-2)
48      mv   t2, t1    # fib(n-2) = fib(n-1)
49      mv   t1, t0    # fib(n-1) = current
50      addi t3, t3, 1
51      ble  t3, a0, loop # for (i = 2; i <= n; i++)
52      mv   a0, t0
53  finish:
54      ret
```

Exemplo 1 - Compilando

No terminal, dê o comando abaixo após entrar em RS5/app/fibonacci-asm:

➤ **make**

Gerando o seguinte retorno:

Assembling src/fibonacci.S...

Linking fibonacci-asm.elf...

/tools/opensource/....warning: fibonacci-asm.elf has a LOAD segment with RWX permissions

Generating fibonacci-asm.bin...

Generating fibonacci-asm.lst...

O comando "make", compila o assembly (fibonacci.S) e gera 4 arquivos:

- **elf** → formato padrão para binário
- **bin** → binário (usado pelo test bech)
- **lst** → contém o código binário e assembly em formato texto
- **map**

Ao lado temos a imagem do arquivo **lst**, que apresenta o programa que utiliza 70_{16} (122_{10}) bytes / 4 = 28: 28 linhas de código.

Disassembly of section .init:

00000000 <_start>:

```
0: 30001073      csrw mstatus,zero
4: 30401073      csrw mie,zero
8: 34401073      csrw mip,zero
c: 00010137      lui sp,0x10
10: 010000ef      jal 20 <main>
```

00000014 <.end>:

```
14: 800002b7      lui t0,0x80000
18: 0002a023      sw zero,0(t0) # 80000000 <__global_pointer$+0x7ffff78c>
1c: ff9ff06f      j 14 <.end>
```

Disassembly of section .text:

00000020 <main>:

```
20: ffc10113      addi sp,sp,-4 # fffc <__global_pointer$+0xf788>
24: 00112023      sw ra,0(sp)
28: 00500513      li a0,5
2c: 018000ef      jal 44 <fibonacci>
30: 00012083      lw ra,0(sp)
34: 00410113      addi sp,sp,4
38: 800022b7      lui t0,0x80002
3c: 00a2a023      sw a0,0(t0) # 80002000 <__global_pointer$+0x8000178c>
40: 00008067      ret
```

00000044 <fibonacci>:

```
44: 00100313      li t1,1
48: 02a35463      bge t1,a0,70 <finish>
4c: 00000293      li t0,0
50: 00000393      li t2,0
54: 00200e13      li t3,2
```

00000058 <loop>:

```
58: 007302b3      add t0,t1,t2
5c: 00030393      mv t2,t1
60: 00028313      mv t1,t0
64: 001e0e13      addi t3,t3,1
68: ffc558e3      bge a0,t3,58 <loop>
6c: 00028513      mv a0,t0
```

00000070 <finish>:

```
70: 00008067      ret
```

Exemplo 1 - Simulação

O RS5 conta com um arquivo testbench, que, como o próprio nome indica, serve para simularmos sinais de entrada em nosso projeto e acompanhar qual saída será gerada.

Os sinais que forem habilitados(1) ou desabilitados(0) no testbench irão sobrescrever os valores dos sinais declarados inicialmente dentro dos seus módulos. Sendo propagados a medida que foram conectados.

Tendo isso em mente:

- 1) Entre em RS5/sim e no arquivo [testbench.sv](#), na linha 59 e aponte o caminho do arquivo binário pertencente ao arquivo anteriormente compilado.
 - o Dê: `localparam string BIN_FILE = "../app/riscv-tests/test.bin";`
 - o Para: `localparam string BIN_FILE = "../app/fibonacci-asm/fibonacci-asm.bin";`
- 2) No terminal, dentro de RS5/sim de o comando "make", obtendo o retorno abaixo no terminal.

```
-- VERILATE -----  
make[1]: .....  
-- RUN -----  
5  
#           710 END OF SIMULATION  
- testbench.sv:287: Verilog $finish  
- Simulation Report: Verilator 5.030 2024-10-27  
- Verilator: $finish at 715ns; walltime 0.006 s; speed 269.100 us/s  
- Verilator: cpu 0.003 s on 1 threads; allocated 57 MB  
-- DONE -----
```

Observação: É necessário ter o [Verilator](#) instalado em seu computador antes de rodar o comando make.

No arquivo [RS5/sim/Makefile](#) nós o utilizamos para a simulação que será apresentada no terminal.



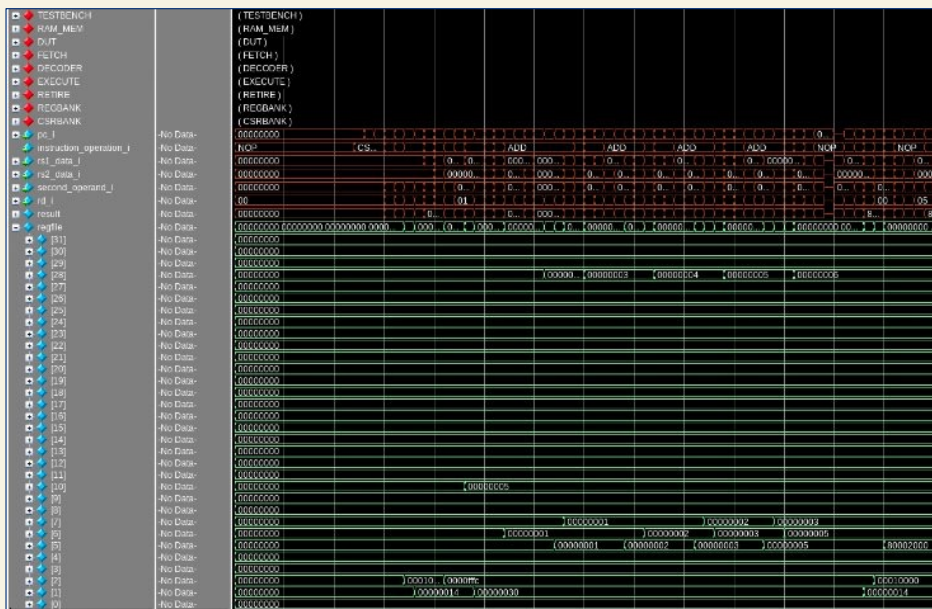
Exemplo 1 - Simulação

Podemos utilizar o Questa para rodar nossa simulação:

Observação: é necessário instalar o [Questa](#) primeiro.

Em seguida, entre na pasta RS5/sim e de o comando:

➤ `vsim -do sim.do &`



```
# ** Warning: (vsim-8315) No condition is true in the unique/priority if/case statement.
# Time: 570 ns Iteration: 5 Process:
/testbench/dut/fetch1/gen_compressed_on/decompresser/#implicit#unique__119 File:
../rtl/decompresser.sv Line: 119
# 5
# 710 END OF SIMULATION
# ** Note: $finish : testbench.sv(290)
# Time: 710 ns Iteration: 1 Instance: /testbench
# End time: 10:08:20 on Jan 08,2025, Elapsed time: 0:00:01
# Errors: 0, Warnings: 1
```

Exemplo 1 - Profiling

Vamos analisar alguns dados do sistema, para isso é necessário conferir se dois sinais existentes em nosso arquivo `testbench.sv` estão **habilitados** (1).

Portanto, em RS5/sim no arquivo `testbench.sv` e verifique:

Linha 47: `localparam bit USE_HPMCOUNTER = 1'b1;`

Linha 52: `localparam bit PROFILING = 1'b1;`

Caso alguma destas variáveis estejam recebendo o bit "0" (**1'b0**), então as altere para que recebam "1", como no exemplo acima.

Já mais abaixo, podemos encontrar o local/arquivo, para onde os dados obtidos estão sendo enviados:

Linha 55: `localparam string PROFILING_FILE = './results/Report.txt';`

Sendo assim, reexecute a simulação e digite no terminal o comando abaixo para abrir o arquivo, e compare com o resultado ao lado:

➤ `more results/Report.txt`

Clock Cycles:	62
Instructions Retired:	43
Instructions Compressed:	2
Instructions Killed:	18
Context Switches:	2
Exceptions Raised:	0
Interrupts Acked:	0
Misaligned Jumps:	0
CYCLES WITH::	
HAZARDS:	0
STALL:	0
BUBBLES (INC. HAZARDS):	19
INSTRUCTION COUNTERS:	
LUI_SLT:	3
LOGIC:	0
ADDSUB:	24
SHIFT:	0
BRANCH:	5
JUMP:	4
LOAD:	1
STORE:	3
SYS:	0
CSR:	3
MUL:	0
DIV:	0

Endereços de memória mapeados no TB (testebench)

fibonacci.S

```
22 .section .text
23 main:
24     addi sp, sp, -4
25     sw   ra, 0(sp)
26
27     li   a0, 5
28     jal  fibonacci
29
30     lw   ra, 0(sp)
31     addi sp, sp, 4
32
33     li   t0, 0x80002000
34     sw   a0, 0(t0)
35
36     ret
```

```
12
13 # Set stack to top of data memory
14 li   sp, MEM_SIZE
15
16 jal main
17 .end:
18 li   t0, 0x80000000
19 sw   zero, 0(t0) # End simulation
20 j    .end
```

Impressão de Inteiro

Final de Simulação

testbench.sv

```
280 //////////////////////////////////////////////////
281 // Memory Mapped regs
282 //////////////////////////////////////////////////
283 int fd;
284 initial begin
285     fd = $fopen(OUTPUT_FILE,"w");
286 end
287
288 always_ff @(posedge clk) begin
289     if (enable_tb) begin
290         // OUTPUT REG
291         if ((mem_address == 32'h80004000 || mem_address == 32'h80001000) && mem_write_enable != '0) begin
292             char <= mem_data_write[7:0];
293             $write("%c",char);
294             if (char != 8'h00)
295                 $fwrite(fd,"%c",char);
296             $fflush();
297         end
298         else if (mem_address == 32'h80002000 && mem_write_enable != '0) begin
299             $write(" %0d\n",mem_data_write);
300             $fwrite(fd,"%0d\n",mem_data_write);
301             $fflush();
302         end
303         // END REG
304         if (mem_address == 32'h80000000 && mem_write_enable != '0) begin
305             $display(" \n# %0t END OF SIMULATION", $time);
306             $fdisplay(fd," \n# %0t END OF SIMULATION", $time);
307             $finish;
308         end
309     end
310     else begin
311         data_tb <= '0;
312     end
313 end
314
315 endmodule
```

Exemplo 2 - C + Assembly inline

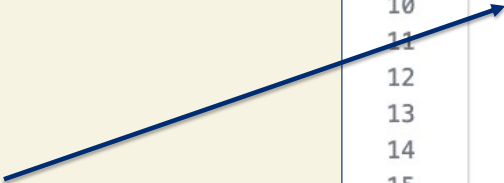
O segundo exemplo ilustra:

- código em C
- funções em assembly inline
- Inicialização do processador em assembly

Arquivo crt0.s – C runtime 0

- Este arquivo chama função C

```
app > fibonacci-c-inline > src > ASM crt0.S
1  .section .init
2  .global _start
3  ∨ _start:
4      # Configure system status MPP=0, MPIE=0, MIE=0
5      csrw    mstatus, zero
6
7      # Mask interrupts
8      csrw    mie, zero
9
10     # Clear pending interrupts
11     csrw    mip, zero
12
13     # Set stack to top of data memory
14     li      sp, MEM_SIZE
15
16     jal main
17  ∨ .end:
18     li      t0, 0x80000000
19     sw      zero, 0(t0)    # End simulation
20     j       .end
21
```



Exemplo 2 - C + Assembly inline

app > fibonacci-c-inline > src > C fibonacci.c > PRINTI

```
1  #define PRINTI (*((volatile unsigned int*)0x80002000))
2
3  int fibonacci(int n)
4  {
5      if (n <= 1)
6          return n;
7
8      int current = 0;
9      int prev1 = 1;
10     int prev2 = 0;
11
12     for (int i = 2; i <= n; i++) {
13         __asm__ (
14             "add %0, %1, %2\n"
15             "mv %2, %1\n"
16             "mv %1, %0\n"
17             : "+r"(current), "+r"(prev1), "+r"(prev2)
18             :
19             :
20         );
21     }
22
23     return current;
24 }
25
26 void main()
27 {
28     volatile int value = 5;
29     int ret = fibonacci(value);
30     PRINTI = ret;
31     return;
32 }
```

Endereço mapeado em memória para receber resultado.

Definição de variáveis.

Início do assembly.

Código assembly, mas com registradores escolhidos pelo compilador.

Ligação variável-registrador.

Exemplo 2 - Compilando

Abra seu terminal em

➤ RS5/app/fibonacci-c-inline

E dê o comando:

➤ make

Obtendo o retorno abaixo:

Compiling src/fibonacci.c...

Assembling src/crt0.S...

Linking fibonacci-c-inline.elf...

/tools/opensource/riscv64-elf/.../bin/ld: warning: fibonacci-c-inline.elf has a LOAD segment with RWX permissions

Generating fibonacci-c-inline.bin...

Generating fibonacci-c-inline.lst...

Exemplo 2 - Compilando

Analisando os registradores utilizados pelo compilador

fibonacci-c-inline.c

```
for (int i = 2; i <= n; i++) {  
    __asm__ (  
        "add %0, %1, %2\n"  
        "mv  %2, %1\n"  
        "mv  %1, %0\n"  
        : "+r"(current), "+r"(prev1), "+r"(prev2)  
        :  
        :  
    );  
}
```

fibonacci-c-inline.lst

```
0000004c <fibonacci>:  
4c: 00100793      li  a5,1  
50: 00050693      mv  a3,a0  
54: 02a7d463      bge a5,a0,7c <fibonacci+0x30>  
58: 00200613      li  a2,2  
5c: 00000713      li  a4,0  
60: 00000513      li  a0,0  
64: 00160613      addi a2,a2,1  
68: 00e78533      add a0,a5,a4  
6c: 00078713      mv  a4,a5  
70: 00050793      mv  a5,a0  
74: fec6d8e3      bge a3,a2,64 <fibonacci+0x18>  
78: 00008067      ret  
7c: 00008067      ret
```

Exemplo 2 - Simulando

No seu terminal entre em RS5/sim

- Modifique o `testbench.sv`

```
localparam string    BIN_FILE    = "../app/fibonacci-c-inline/fibonacci-c-inline.bin";
```

- No terminal execute `make`

```
-- VERILATE -----
```

```
make[1]: Entering directory '/userdata/home-inf-el8/instructors/fernando.moraes/RS5/sim/obj_dir'
```

```
....
```

```
-- RUN -----
```

```
5
```

```
#          740 END OF SIMULATION
```

```
- testbench.sv:287: Verilog $finish
```

```
- Simulation Report: Verilator 5.030 2024-10-27
```

```
- Verilator: $finish at 745ns; walltime 0.006 s; speed 245.936 us/s
```

```
- Verilator: cpu 0.003 s on 1 threads; allocated 57 MB
```

```
-- DONE -----
```

Exemplo 2 - Profiling

Assembly

Clock Cycles:	62
Instructions Retired:	43
Instructions Compressed:	2
Instructions Killed:	18
Context Switches:	2
Exceptions Raised:	0
Interrupts Acked:	0
Misaligned Jumps:	0
CYCLES WITH::	
HAZARDS:	0
STALL:	0
BUBBLES (INC. HAZARDS):	19
INSTRUCTION COUNTERS:	
LUI_SLT:	3
LOGIC:	0
ADDSUB:	24
SHIFT:	0
BRANCH:	5
JUMP:	4
LOAD:	1
STORE:	3
SYS:	0
CSR:	3
MUL:	0
DIV:	0

C inline

Clock Cycles:	65
Instructions Retired:	45
Instructions Compressed:	1
Instructions Killed:	18
Context Switches:	2
Exceptions Raised:	0
Interrupts Acked:	0
Misaligned Jumps:	0
CYCLES WITH::	
HAZARDS:	1
STALL:	0
BUBBLES (INC. HAZARDS):	20
INSTRUCTION COUNTERS:	
LUI_SLT:	3
LOGIC:	0
ADDSUB:	24
SHIFT:	0
BRANCH:	5
JUMP:	4
LOAD:	2
STORE:	4
SYS:	0
CSR:	3
MUL:	0
DIV:	0

Exemplo 3 - C + newlib

newlib

- implementação da biblioteca C para embarcados
- suporte a funções como malloc, printf, strcpy.....
- contém o crt0

fonte

- Apenas o código C



Escrita mapeada em memória
abstraída pelo printf



```
app > fibonacci-c-newlib > src > C fibonacci.c > ...
1  #include <stdio.h>
2
3  int fibonacci(int n)
4  {
5      if (n <= 1)
6          return n;
7
8      int current = 0;
9      int prev1 = 1;
10     int prev2 = 0;
11
12     for (int i = 2; i <= n; i++) {
13         current = prev1 + prev2;
14         prev2 = prev1;
15         prev1 = current;
16     }
17
18     return current;
19 }
20
21 int main()
22 {
23     volatile int value = 5;
24     int ret = fibonacci(value);
25     printf("%d\n", ret);
26     return 0;
27 }
```

Exemplo 3 - Compilando

No terminal em RS5/app/fibonacci-c-newlib execute o comando **make**

Obtendo como resultado:

Compiling src/fibonacci.c...

Compiling ../common/newlib.c...

Assembling ../common/crt0.S...

Linking fibonacci-c-newlib.elf...

/tools/opensource/.....lf/bin/ld: warning: fibonacci-c-newlib.elf has a LOAD segment with RWX permissions

Generating fibonacci-c-newlib.bin...

Generating fibonacci-c-newlib.lst...

Exemplo 3 - Compilando

1) No RS5/sim/testbench.sv

- localparam string BIN_FILE = "../app/fibonacci-c-newlib/fibonacci-c-newlib.bin";

2) make

-- VERILATE -----

...

V e r i l a t i o n R e p o r t: Verilator 5.030 2024-10-27 rev v5.030 (mod)

- Verilator: Built from 0.396 MB sources in 33 modules, into 0.759 MB in 9 C++ files needing 0.001 MB

- Verilator: Walltime 11.966 s (elab=0.105, cvt=0.301, bld=10.714); cpu 1.159 s on 8 threads; allocated 38.316 MB

-- RUN -----

5# 57220 END OF SIMULATION

- testbench.sv:287: Verilog \$finish

- S i m u l a t i o n R e p o r t: Verilator 5.030 2024-10-27

- Verilator: \$finish at 57us; walltime 0.013 s; speed 5.336 ms/s

- Verilator: cpu 0.011 s on 1 threads; allocated 57 MB

-- DONE -----

Exemplo 3 - Profiling

Assembly

Clock Cycles:	62
Instructions Retired:	43
Instructions Compressed:	2
Instructions Killed:	18
Context Switches:	2
Exceptions Raised:	0
Interrupts Acked:	0
Misaligned Jumps:	0
CYCLES WITH::	
HAZARDS:	0
STALL:	0
BUBBLES (INC. HAZARDS):	19
INSTRUCTION COUNTERS:	
LUI_SLT:	3
LOGIC:	0
ADDSUB:	24
SHIFT:	0
BRANCH:	5
JUMP:	4
LOAD:	1
STORE:	3
SYS:	0
CSR:	3
MUL:	0
DIV:	0

C inline

Clock Cycles:	65
Instructions Retired:	45
Instructions Compressed:	1
Instructions Killed:	18
Context Switches:	2
Exceptions Raised:	0
Interrupts Acked:	0
Misaligned Jumps:	0
CYCLES WITH::	
HAZARDS:	1
STALL:	0
BUBBLES (INC. HAZARDS):	20
INSTRUCTION COUNTERS:	
LUI_SLT:	3
LOGIC:	0
ADDSUB:	24
SHIFT:	0
BRANCH:	5
JUMP:	4
LOAD:	2
STORE:	4
SYS:	0
CSR:	3
MUL:	0
DIV:	0

C + newlib

Clock Cycles:	5738
Instructions Retired:	3760
Instructions Compressed:	3
Instructions Killed:	1808
Context Switches:	159
Exceptions Raised:	0
Interrupts Acked:	0
Misaligned Jumps:	0
CYCLES WITH::	
HAZARDS:	99
STALL:	0
BUBBLES (INC. HAZARDS):	1908
INSTRUCTION COUNTERS:	
LUI_SLT:	34
LOGIC:	52
ADDSUB:	1584
SHIFT:	7
BRANCH:	755
JUMP:	184
LOAD:	323
STORE:	814
SYS:	0
CSR:	3
MUL:	0
DIV:	2

Exercício

Aplicação: copie a pasta hello para uma nova pasta, por exemplo “exercício”.

Considere três strings, por exemplo:

- `const char str1[] = "Texto exemplo para Curso";`
- `const char str2[] = "Formacao de Cozinheiros Experts";`
- `const char str3[] = "Chips para Microeletronica";`

Incluir no exercício bibliotecas como (sugestão):

- `#include <stdio.h>`
- `#include <stdlib.h>`
- `#include <string.h>`

O exercício deve:

- Obter uma substring de `str1` correspondente à última palavra (dica função `strchr`)
- Obter uma substring de `str2` correspondente à primeira palavra (dica função `strtok`)
- Obter uma substring de `str3` correspondente à última palavra (dica função `strchr`)
- Usar `malloc` para alocar uma nova `string (result)` correspondente aos tamanhos das substrings
- Concatenar as três substrings em `result` e imprimir

Exercício

- Compile o programa na pasta em que foi criado, usando `make`
- Altere o `testbench.sv`, como nos exemplos anteriores.
- Dê `make` na pasta `sim`.

-- VERILATE -----

make[1]: Entering directory '/userdata/home-inf-el8/instructors/fernando.moraes/RS5/sim/obj_dir'

make[1]: Nothing to be done for 'default'.

make[1]: Leaving directory '/userdata/home-inf-el8/instructors/fernando.moraes/RS5/sim/obj_dir'

- V e r i l a t i o n R e p o r t: Verilator 5.030 2024-10-27 rev v5.030 (mod)

- Verilator: Built from 0.000 MB sources in 0 modules, into 0.000 MB in 0 C++ files needing 0.000 MB

- Verilator: Walltime 0.116 s (elab=0.000, cvt=0.000, bld=0.045); cpu 0.007 s on 8 threads; allocated 7.465 MB

-- RUN -----

Substring 1: Curso

Substring 2: Formacao

Substring 3: Microeletronica

String concatenada: Curso Formacao Microeletronica

196910 END OF SIMULATION

- testbench.sv:287: Verilog \$finish

- S i m u l a t i o n R e p o r t: Verilator 5.030 2024-10-27

- Verilator: \$finish at 169us; walltime 0.030 s; speed 6.946 ms/s

- Verilator: cpu 0.028 s on 1 threads; allocated 57 MB

-- DONE -----

Exercício

Usando questa, abra o terminal e execute:

```
vsim -c -do sim.do
```

****Observação:** -c significa questa sem interface gráfica)

```
# Loading work.lrsc(fast)
# Loading work.amo(fast)
# Loading work.retire(fast)
# Loading work.CSRBank(fast)
# Loading work.RAM_mem(fast)
# Loading work.plic(fast)
# Loading work.rtc(fast)
# Substring 1: Curso
# Substring 2: Formacao
# Substring 3: Microeletronica
# String concatenada: Curso Formacao Microeletronica #
196910 END OF SIMULATION
# ** Note: $finish : testbench.sv(287)
# Time: 196910 ns Iteration: 1 Instance: /testbench
# 1
# Break in Module testbench at testbench.sv line 287
```

Exercício - Profiling

Clock Cycles	19682
Instructions Retired	12297
Instructions Compressed	3
Instructions Killed	6420
Context Switches	1050
Exceptions Raised:	0
Interrupts Acked	0
Misaligned Jumps	0
CYCLES WITH::	
HAZARDS	
STALL	0
BUBBLES (INC. HAZARDS):	7385
INSTRUCTION COUNTERS:	
LUI_SLT	88
LOGIC	185
ADDSUB	4384
SHIFT	13
BRANCH	2525
JUMP	992
LOAD	2103
STORE	2002
SYS	0
CSR	3
MUL	0
DIV	0



Fim tutorial 1

RS5

Fernando Gehm Moraes - fernando.moraes@pucrs.br

Willian Analdo Nunes - willian.nunes@edu.pucrs.br

Angelo Dal Zotto - angelo.dalzotto@edu.pucrs.br