1) Diferencie ISAs Modulares de ISAs incrementais.

ISA incremental: novos processadores devem implementar não apenas novas extensões ISA, mas também todas as extensões do passado. Segundo exemplo do livro seria como se: um restaurante servisse apenas uma refeição a preço fixo, que inicia por um pequeno jantar de hambúrguer e milkshake. Com o tempo, acrescenta-se batatas fritas e depois um sundae de sorvete, seguido de salada, torta, vinho, massa vegetariana, bife, cerveja, ad infinitum até se tornar um grande banquete.

ISA modular: No núcleo há um ISA básico, chamado RV32I, que executa uma pilha completa de software. O RV32I está congelado e nunca será alterado, o que dá aos criadores de compiladores, desenvolvedores de sistemas operacionais e programadores de linguagem assembly um destino estável. A modularidade vem de extensões padrão opcionais que o hardware pode incluir ou não. Ou seja, o chef precisa cozinhar apenas o que os clientes querem—não um banquete para cada refeição—e os clientes pagam apenas pelo que pedem.

2) Cite e descreva sucintamente as 7 medidas relativas ao projeto de uma ISA.

1. custo (ícone da moeda de dólar)
2. simplicidade (roda)
3. desempenho (velocímetro)
4. isolamento de arquitetura da implementação (metades separadas de um círculo)
5. espaço para crescimento (acordeão)
6. tamanho do programa (setas opostas)
7. facilidade de programação / compilação / “linkagem” (“tão fácil quanto o ABC”).

1- Os processadores são implementados como circuitos integrados, comumente chamados de chips ou dies(recortes), e a fabricação de silício resulta em pequenas falhas espalhadas sobre o wafer, então quanto menor a área, menor a fração que será defeituosa. Um arquiteto deseja manter a ISA simples para reduzir o tamanho dos processadores que o implementam

2- Dada a sensibilidade do custo à complexidade, os arquitetos desejam uma ISA simples para reduzir a área de impressão. A simplicidade também reduz o custo da documentação e a dificuldade de fazer com que os clientes entendam como usar o ISA. Ironicamente, as instruções simples são muito mais prováveis de serem usadas do que as complexas.

3- O desempenho pode ser considerado em três termos: *instruções/programa × media ciclos clock/instrução × tempo/ciclo clock = tempo/programa*. Mesmo que uma ISA simples possa executar mais instruções por programa do que uma ISA complexa, ela pode compensar isso com um ciclo de clock mais rápido ou uma média menor de ciclos de clock por instrução (CPI).

4- A distinção original entre arquitetura e implementação, que remete à década de 1960, é que arquitetura é o que um programador de linguagem assembly precisa saber para escrever um programa correto, mas sem se preocupar com o desempenho desse programa. Enquanto arquitetos não devem colocar recursos que ajudam apenas uma implementação em um determinado momento, esses também não devem colocar features que complicam algumas implementações.

5- Com o fim da Lei de Moore, o único caminho a seguir para grandes melhorias no custo-desempenho é adicionar instruções personalizadas para domínios específicos, como deep learning, realidade aumentada, otimização combinatória, gráficos e assim por diante. Isso significa que é importante hoje que uma ISA reserve espaço de opcode para futuras melhorias.

6- Quanto menor o programa, menor a área necessária em um chip para a memória do programa, o que pode representar um custo significativo para dispositivos embarcados.

7- Como os dados em um registrador são muito mais rápidos de serem acessados do que os dados na memória, é importante que os compiladores realizem um bom trabalho na alocação de registradores. Essa tarefa é muito mais fácil quando há muitos registradores em vez de poucos.Mais registradores certamente facilitam a vida de compiladores e programadores de linguagem assembly.

3) Quais são as 6 ações (estágios) a serem realizadas em uma chamada de função? Descreva em suas palavras o que significa cada uma destas ações quando se utiliza o RISC-V.

1. Colocar os argumentos onde a função possa acessá-los.

2. Saltar para a função (utilizando a instrução jal do RV32I's).

3. Adquirir recursos de armazenamento local que a função necessita, salvando registradores conforme necessário.

4. Executar a tarefa desejada da função.

5. Colocar o valor do resultado da função onde o programa de chamada pode acessá-lo, restaurar qualquer registrador, e liberar quaisquer recursos de armazenamento local.

6. Como uma função pode ser chamada de vários pontos em um programa, retornar o controle para seu respectivo ponto de origem (utilizando ret).

1 Chamar os dados que serão utilizados da memória - lw, la…
2 Se uma condição for satisfeita o programa deve executar a função especificada
3 Chamar os registradores/endereço de memória necessários para salvar os resultados obtidos quando uma função é executada
4 Executar as operações que são descritas em uma função, multiplicar, somar…
5 Salvar os resultados da função executada nos registradores/ endereço de memória chamados no passo 3
6 Após executar a função o programa deve continuar sendo executado a partir da instrução imediata após a chamada de função.

4) Comente, sob o ponto de vista da performance, a afirmativa:
"Para obter um bom desempenho, tente manter as variáveis nos registradores em vez da memória, mas por outro lado, evite também acessar a memória frequentemente para salvar e restaurar esses valores."
Porque o acesso ao registrador é muito mais rápido do que o acesso à memória, pelo registrador faz-se milhões de acessos a mais do que pela memória, isso acumulado ao longo das operações pode significar segundos ou minutos de delay.

5) O que significa ter os registradores salvos (ou não salvos) durante as chamadas de função? No caso dos registradores salvos: (a) quem é responsável? (b) Onde o conteúdo dos registradores é salvo?
Salvar um registrador significa ter garantia que os dados serão preservados durante a chamada da função
a) o programador é responsável por controlar os registradores
b) na pilha

6) O que são pseudo-instruções? Apresente 5 pseudo-instruções do RISC-V com as respectivas instruções para as quais as mesmas são convertidas?
São configurações inteligentes de instruções regulares, elas costumam ser substituídas pelo **montador** ao gerar instruções para o computador na forma de **Linguagem de Máquina**. Pseudo-Instruções são na verdade combinações de mais de uma instrução.

mv rd, rs	|	*addi rd, rs, 0*
jal offset	|	*jal x1, offset*
jalr rs	|	*jalr x1, rs, 0*
call offset	|	*auipc x1, offset[31:12] jalr x1, x1, offset[11:0]*
fence	|	*fence iorw, iorw*

7) O que são e para que servem as diretivas assembler? Qual a função das seguintes diretivas assembler:
(a) .data
(b) .bss
(c) .word
São comandos específicos para o assembler, e não código para ser traduzido por ele. Estas diretivas informam ao assembler onde colocar código e dados, especificam constantes de código e dados para uso no programa, e assim por diante.
a- Para armazenar variáveis globais
b- Armazena variáveis globais que iniciam com 0
c- Armazena as n quantidade de 32 bits em palavras de memória sucessivas.

8) Qual a função do Linker? O que é e para que serve a tabela de símbolos?
o Linker, ou ligador, permite que arquivos individuais sejam compilados e montados separadamente. Ele “costura” o novo código objeto junto aos módulos de linguagem de máquina existentes, como bibliotecas. O linker verifica se a ABI do programa corresponde a todas as suas bibliotecas.
A tabela de símbolos inclui todos os rótulos no programa que devem receber endereços como parte do processo de vinculação realizado pelo linker. Essa lista inclui rótulos para dados, bem como para código

9) Em relação a alocação de memória:
(a) Quais as regiões de memória e o que fica armazenado em cada uma das regiões?
(b) Qual a direção que os dados são colocados em cada uma das regiões?
(c) Na convenção do RISC-V existem registradores que apontam para cada uma das regiões? Quais?

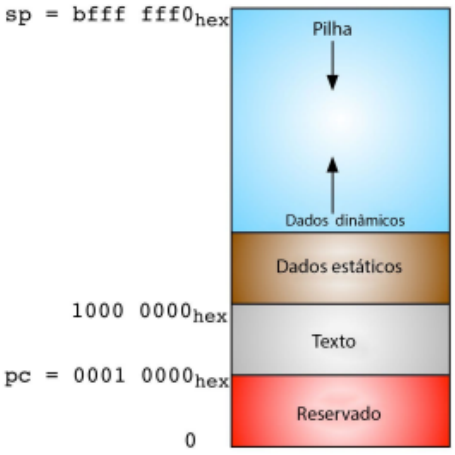

Figura 3.10: Alocação de memória para programa e dados do RV32I. Os endereços superiores estão no topo da figura, enquanto os inferiores estão na parte inferior. Nesta convenção de software do RISC-V, o ponteiro de pilha (sp) inicia em bfff fff0$_{hex}$ e cresce para baixo em direção aos dados estáticos. O *text* (código do programa) inicia em 0001 0000$_{hex}$ e inclui todas bibliotecas linkadas dinamicamente. Os dados estáticos iniciam imediatamente acima da região do text; Neste exemplo, consideramos que o endereço é 1000 0000$_{hex}$. Dados dinâmicos, alocados em C por `malloc()`, estão logo acima dos dados estáticos. Chamado de `heap`, este conjunto cresce em direção à pilha, e inclui todas bibliotecas linkadas dinamicamente.

Como destacado na figura a memória fica dividida em 4 partes, na pilha ficando os dados dinâmicos, que estão mudando, entrando ou saindo da pilha conforme são usados. os Dados estáticos que não sofrem alterações. Text que é o código do programa Assembly, área .text e a área reservada ao program counter.
Em risc-V os seguintes registradores apontam para regiões da memória:

REG	***ABI***	***APONTA***
x2	sp	Ponteiro de pilha
x3	gp	Ponteiro global
x4	tp	Ponteiro de Thread

10) O que é "cross compiling" no processo de compilação do programa?

O linker verifica se a ABI do programa corresponde a todas as suas bibliotecas. Embora o compilador tenha suporte muitas combinações de extensões ISA e ABIs, apenas alguns conjuntos de bibliotecas podem ser instalados. Assim, um erro comum é *vincular um programa sem ter as bibliotecas compatíveis instaladas*. O linker não produzirá uma mensagem de diagnóstico útil nesse caso; ele tentará simplesmente vincular-se a uma biblioteca incompatível e, em seguida, informará a incompatibilidade. Esse erro geralmente ocorre apenas quando *compila-se em um computador para um computador diferente* (cross compiling).

Fontes:

PATTERSON, David *et al*. Guia prático risc-v: atlas de uma arquitetura aberta. 1.0.0. ed. San francisco: Strawberry canyon, 2019. 215 p. v. 1. ISBN 978-0-9992491-1-6.

MANIERO, Antonio. TL; DR. *In*: O que são registradores e qual é o seu funcionamento básico?. Stackoverflow, 6 ago. 2018. Acesso em: 14 jun. 2022. Disponível em: https://pt.stackoverflow.com/questions/320285/o-que-s%C3%A3o-registradores-e-qual-%C3%A9-o-seu-funcionamento-b%C3%A1sico.

https://pt.wikibooks.org/w/index.php?title=Introdu%C3%A7%C3%A3o_%C3%A0_Arquitetura_de_Computadores/As_Pseudo-Instru%C3%A7%C3%B5es&oldid=281434