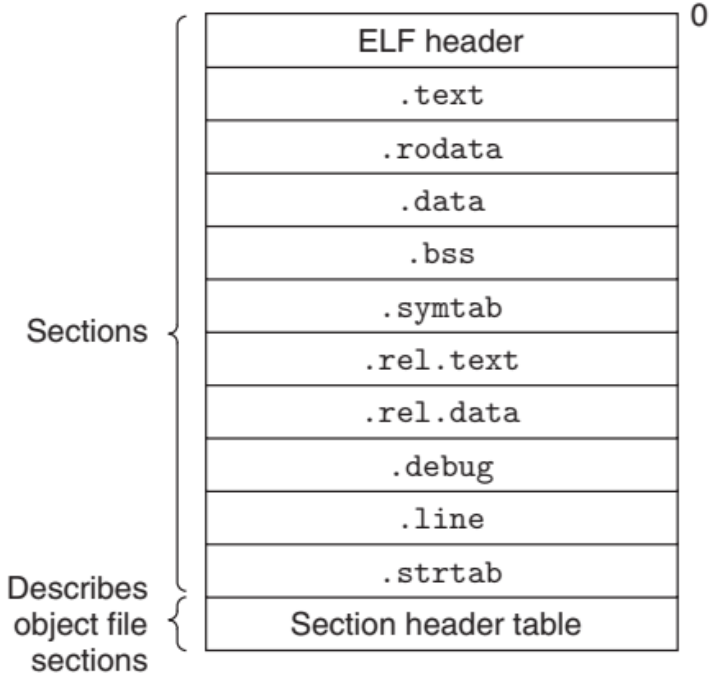
Trabalho 1 – parte 2

- Adicionar as instruções
  - LB, LBU, LH, LHU, SB, SH, SLTI, SLTIU, BGEZ, BLEZ, JAL, JALR, JR
- Programa de teste: t1_P2.asm (*moodle*)
- Dica
  - Para cada instrução a ser adicionada, primeiro entender o seu funcionamento baseado na documentação das instruções (e.g. MIPSISA.pdf no *moodle*) e no simulador MARS
  - Em seguida alterar o código VHDL
  - Simular o nova instrução em VHDL

Trabalho 1 - parte 2

- ELF Object files
  - ELF: Executable Linkable Format
  - Format used by GCC (GNU Compiler Collection)
    - Compilers and assemblers generate linkable object files
    - Linkers generate executable object files



Trabalho 1 – parte 2

- ELF Object files
  - Divided into multiple sections, and some of them are linked by the linker in order to create an executable object file
  - Each section contains information such as code (.text), data (.data, .rodata) and relocation information (.rel.text, .rel.data)

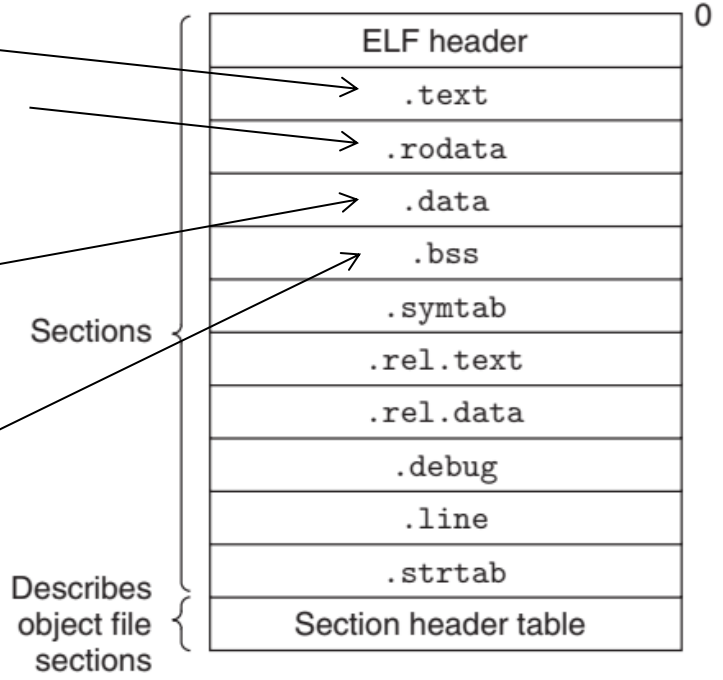
The machine code of the compiled program

Read-only data such as the format strings in *printf* statements

```
printf("Hello world!")
```

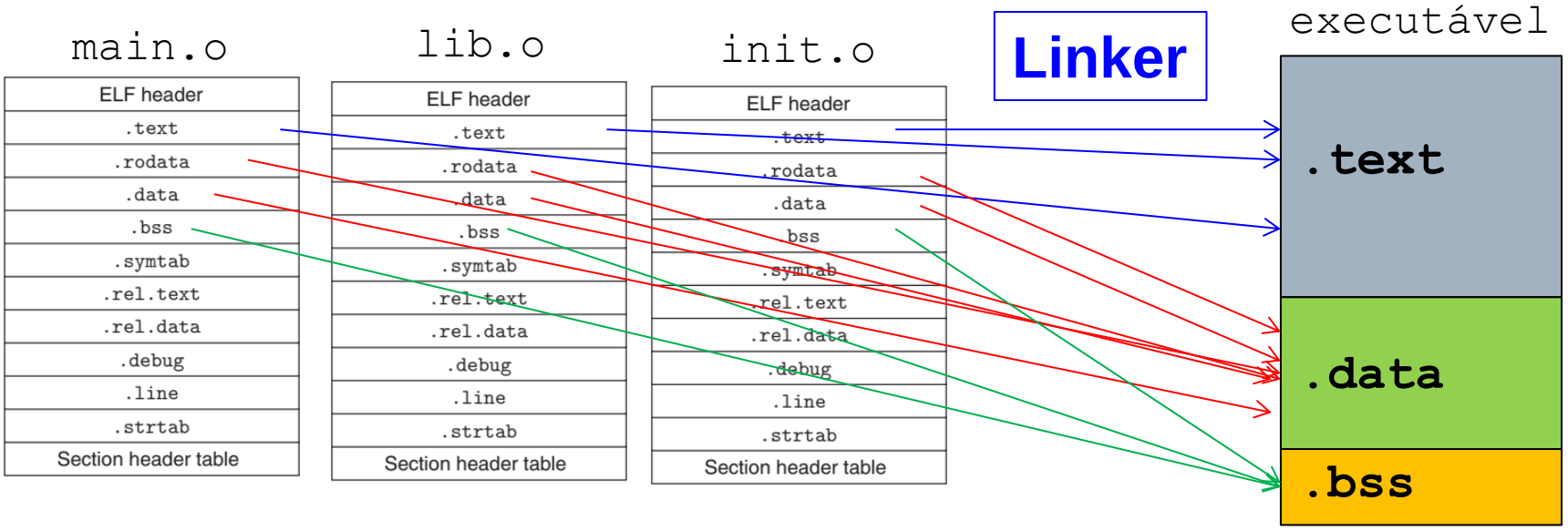
Initialized global/static C variables (`int size = 10`). Local C variables are maintained at run time on the stack, and do not appear on the ELF object file

Uninitialized global/static C variables. Specifies how many bytes are needed for the global and static variables. This section occupies no actual space in the object file.; it is merely a place holder



Trabalho 1 – parte 2

- ELF Object files
  - Divided into multiple sections, and some of them are linked by the linker in order to create an executable object file



Através de um *script* (*linker script*) é possível especificar ao *linker* quais *sections* de quais arquivos de entrada devem ser juntadas em uma *section* do arquivo executável

Trabalho 1 - parte 2

- O objetivo do trabalho é realizar todo o processo de geração de código binário a partir de código em linguagem C
  - Compilação → Montagem → Ligação → Código binário executável
- A primeira coisa a ser feita é o download do IMG **Bare Metal Toolchain** (Windows/Linux)
  - https://codescape.mips.com/components/toolchain/2019.02-04/downloads.html
- Após o download basta descompactar o arquivo

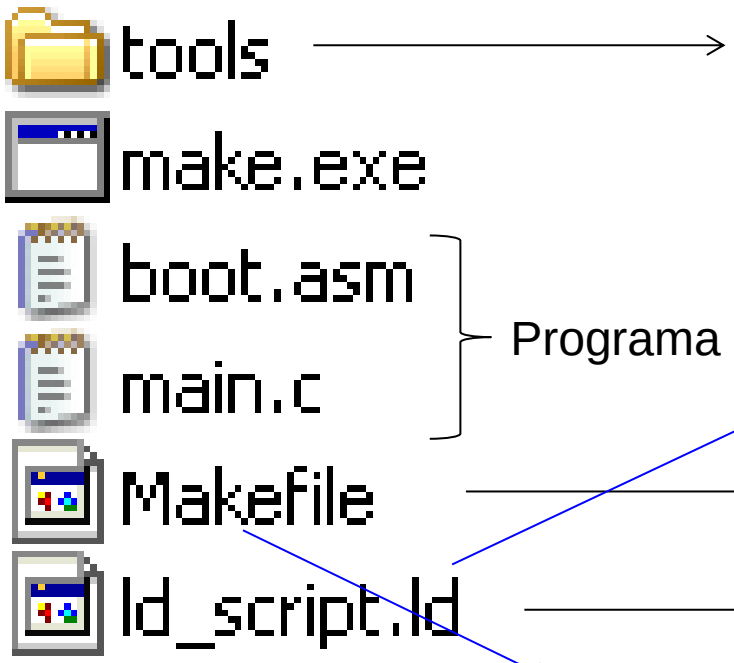
Trabalho 1 - parte 2

- As ferramentas que utilizaremos estão o diretório
  - Codescape.GNU.Tools.Package.2019.02-03.for.MIPS.IMG.Bare.Metal.Windows.x86\mips-img-elf\2019.02-03\bin
    - mips-img-elf-gcc.exe Compilador C
    - mips-img-elf-as.exe Montador
    - mips-img-elf-ld.exe Linker

    - Ferramentas auxiliares
    - mips-img-elf-objdump.exe Geração de *assembly* para *debug*
    - mips-img-elf-objcopy.exe Geração das imagens das memórias
    - mips-img-elf-size.exe Tamanho das seções do executável
- Todo processo de geração de código binário executável será automatizado através de *makefile* e *script* de ligação (*linker script*)

Trabalho 1 - parte 2

- Arquivos para automatizar o processo de geração
  - gcc.zip (*moodle*)



Programa que gera as imagens das memórias de instruções e dados. Pode ser necessário recompilar com *g++*

Programa de exemplo

Setar OFFSET das memórias VHDL e SIZE de acordo com o *ld_script*

script de compilação/montagem

script de ligação

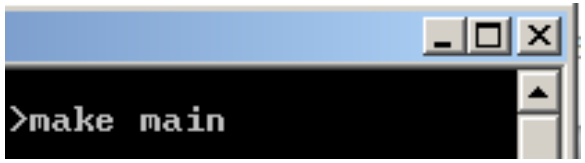
Setar a variável TOOLCHAIN do acordo com a localização do diretório Codescape.GNU.Tools.Package.2019.02-03.for.MIPS.IMG.Bare.Metal.Windows.x86/mips-img-elf/2019.02-03/bin

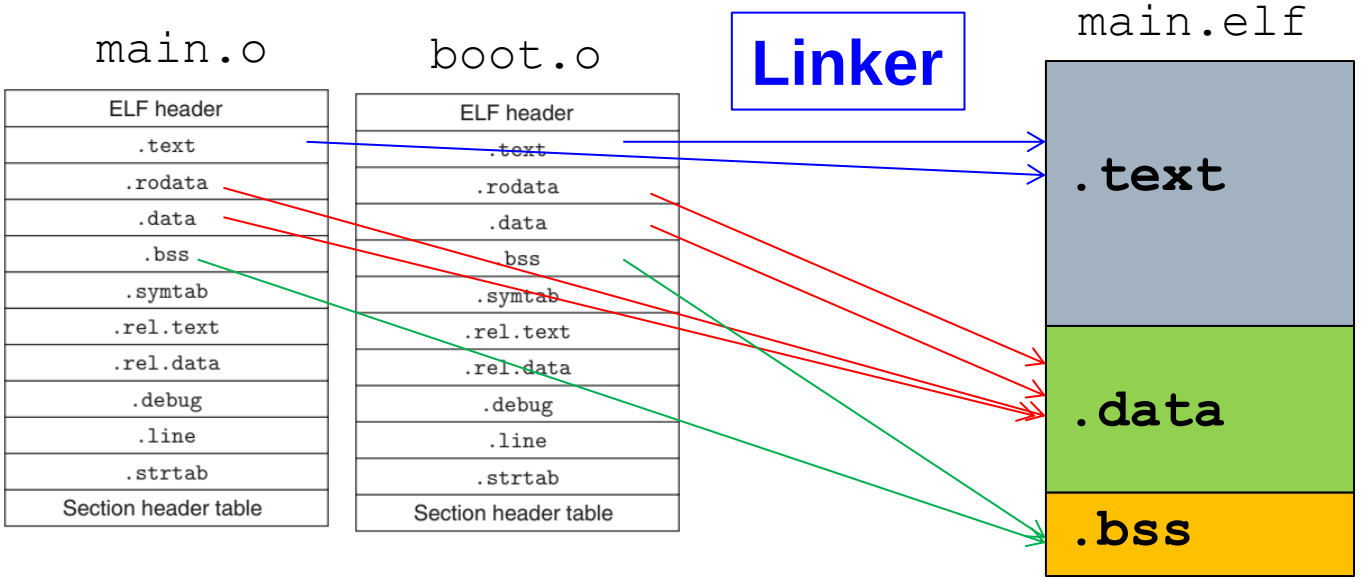
Trabalho 1 - parte 2

- Arquivos gerados pela execução do *makefile*
  - .o: código objetos usado para gerar o executável
  - .elf: código binário executável
  - .lst: código assembly
  - code.bin: imagem da memória de instruções (binário)
  - data.bin: imagem da memóra de dados (binário)
  - code.txt: imagem da memória de instruções (texto)
  - data.txt: imagem da memóra de dados (texto)
  - .map: aquivo de mapeamento nas memórias

Trabalho 1 - parte 2

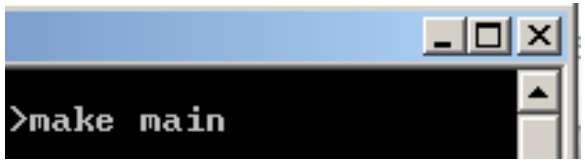
- Exemplo
  - boot.asm: inicializa o *sp* e salta para *main()* (main.c)
  - main.c

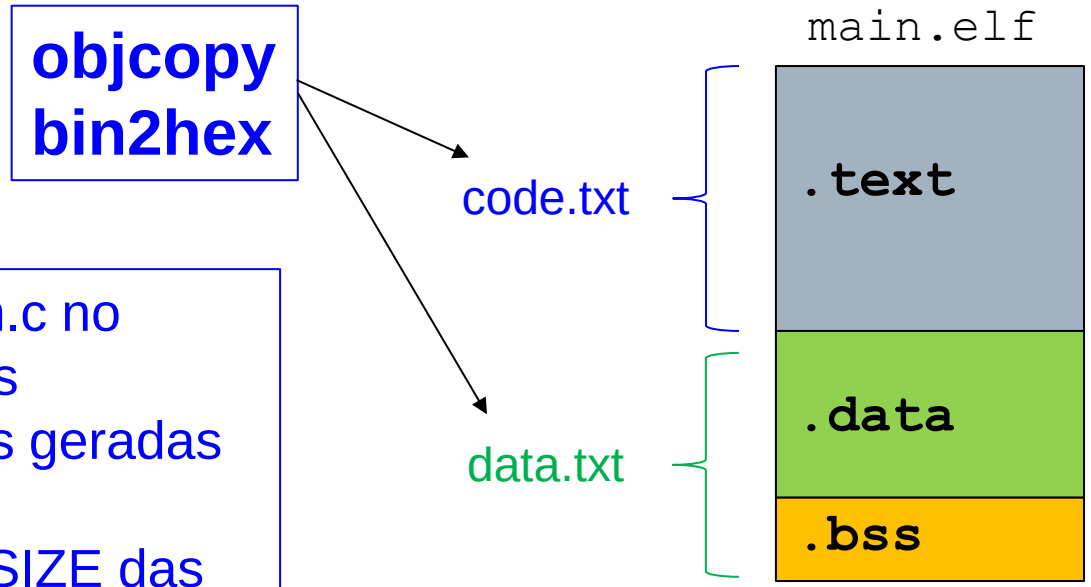




Trabalho 1 - parte 2

- Exemplo
  - boot.asm: inicializa o *sp* e salta para *main()* (main.c)
  - main.c

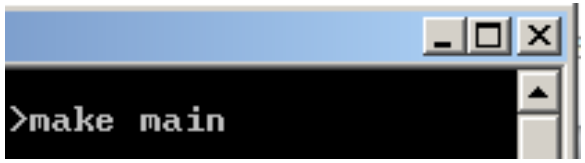




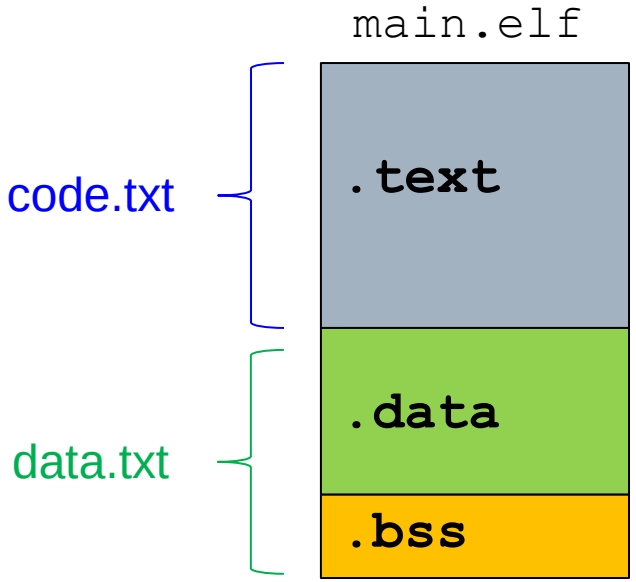
Executar o programa main.c no MIPS VHDL carregando as memórias com as imagens geradas (code.txt e data.txt).
Para este exemplo, setar SIZE das memórias igual a 128

Trabalho 1 - parte 2

- Exemplo
  - boot.asm: inicializa o *sp* e salta para *main()* (main.c)
  - main.c



O progama main.c preenche dois *arrays* na memória dados. Verificar na simulação o preenchimento correto dos *arrays* de acordo com o programa.



Trabalho 1 - parte 2

- Grupos de 2 alunos
  - Apresentação da descrição funcionando com as novas instruções será impreterivelmente dia 5/4
  - A nota do trabalho dará ENORME ÊNFASE à execução correta da simulação
    - Se a simulação não funciona, não há o que apresentar
  - A apresentação será oral, teórico-prática, frente ao computador, onde o grupo deverá explicar ao professor o projeto, a simulação e a implementação
  - Em relação às duvidas, sejam pontuais