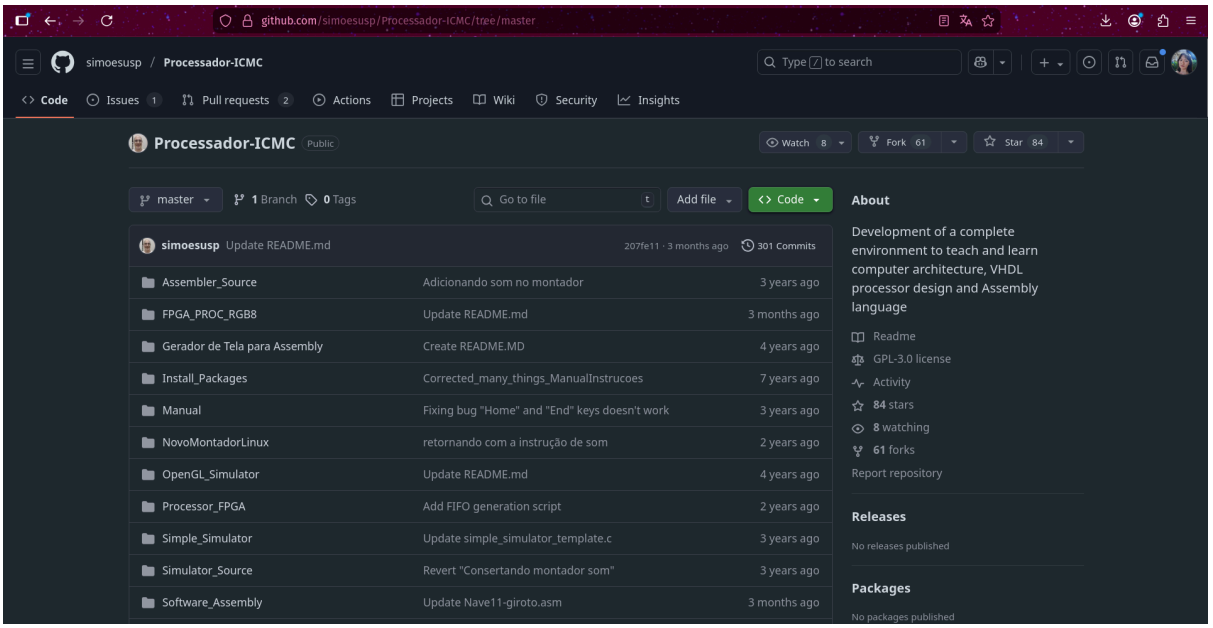
PASSO A PASSO DE COMO IMPLEMENTAR UMA NOVA INSTRUÇÃO

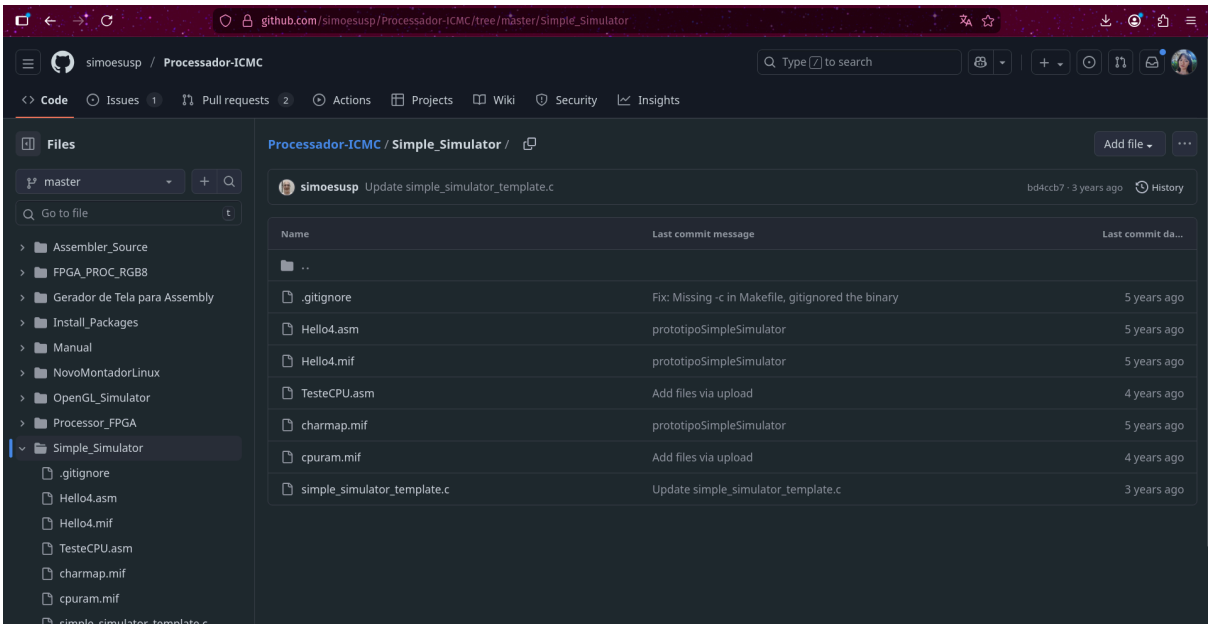
Obs1: esse passo a passo foi feito em um sistema operacional Linux

Passo 1) Achar os arquivos necessários para o trabalho

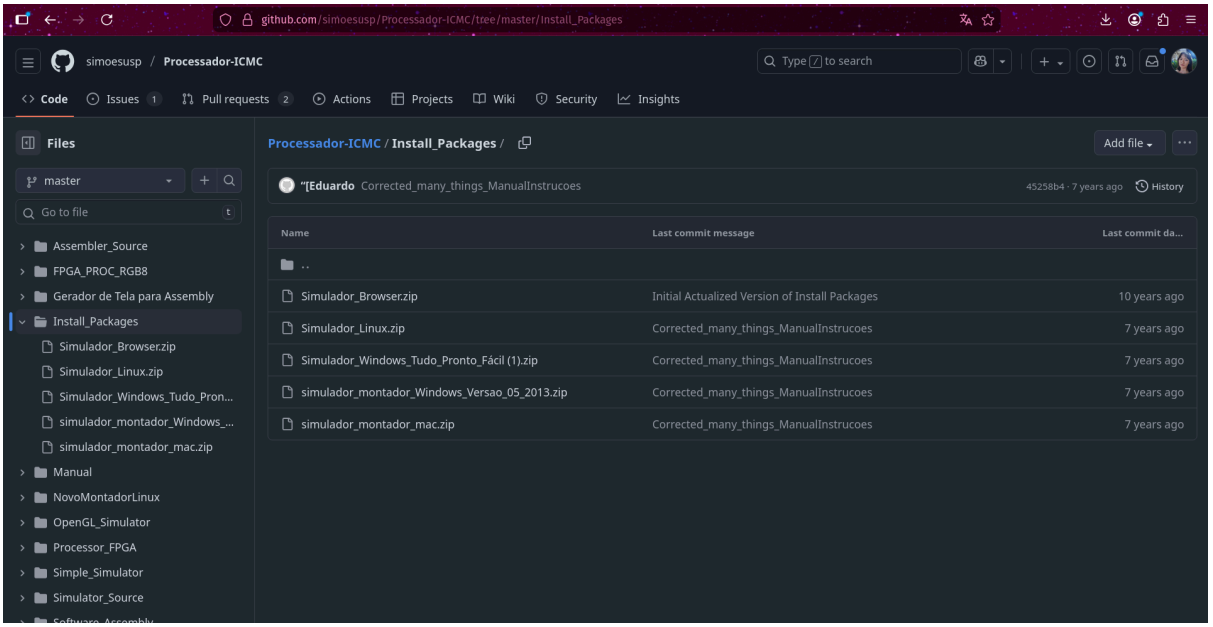
- Ache o repositório Processador-ICMC: https://github.com/simoesusp/Processador-ICMC/tree/master



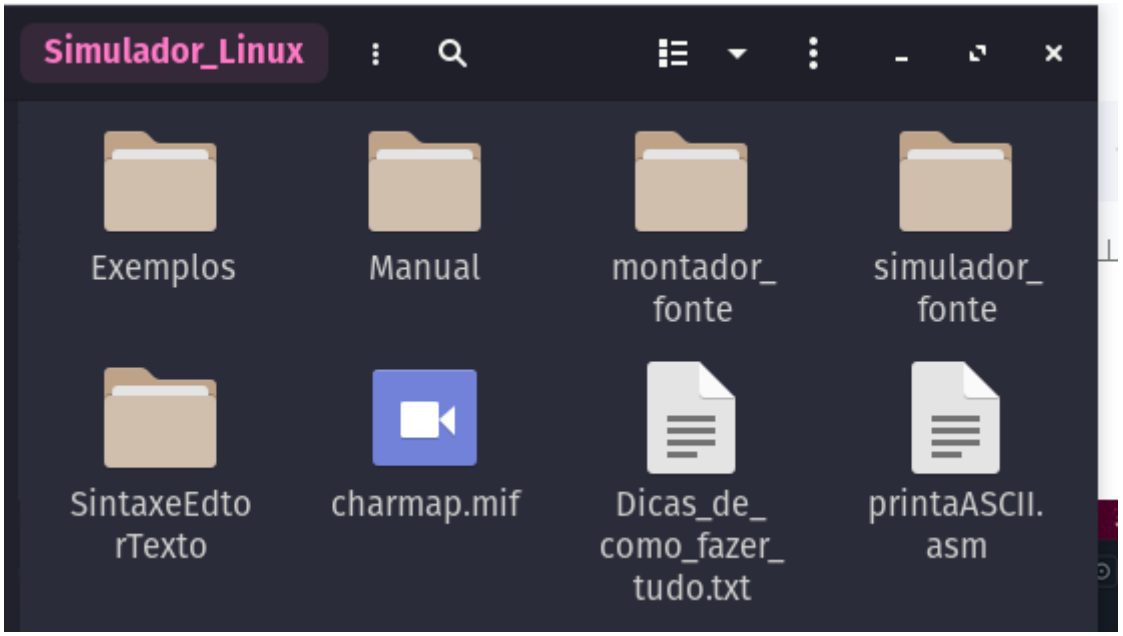
- Vá para a pasta Simple_Simulator e baixe todos esses arquivos. Os arquivos que serão alterados serão o simple_simulator_template.c, TesteCPU.asm e cpuram.mif

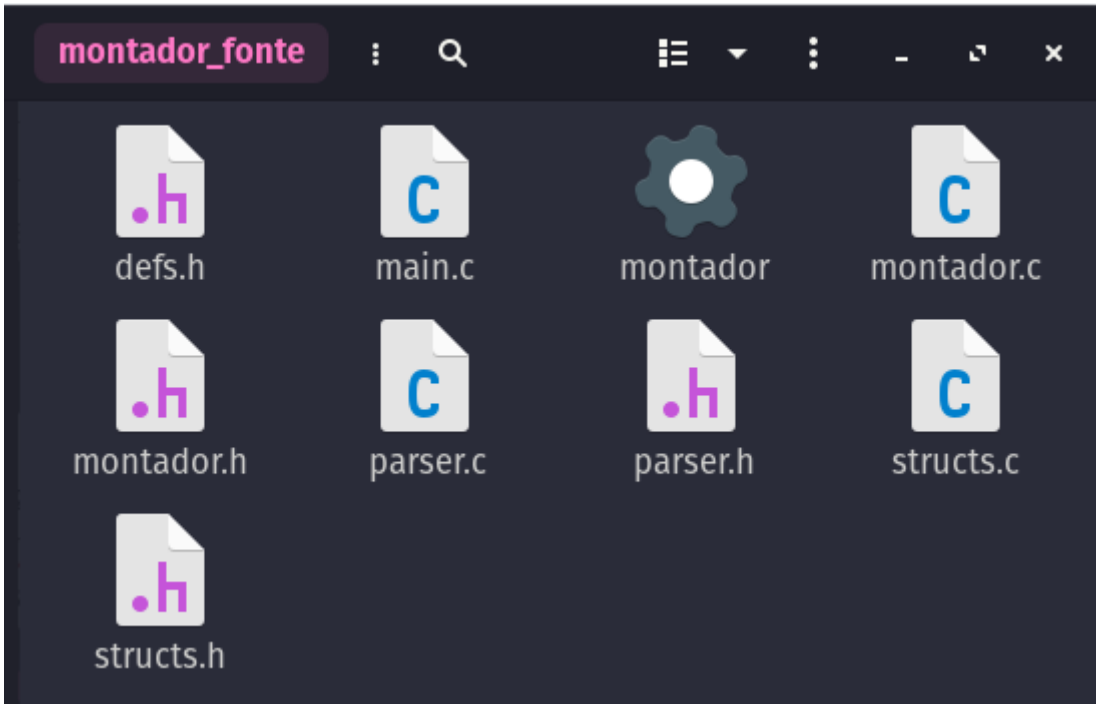


- Volte para a tela anterior e vá para a pasta Install_Packages
- Baixe o zip Simulador_Linux



- Extraia o zip. Para implementar a nova instrução vai ser utilizada os arquivos da pasta montador_fonte (se quiser pode copiar toda essa pasta e os arquivos anteriores baixados em uma pasta a parte para seu trabalho)





Passo 2) Alterações necessárias para a implementação e terminal

- **Obs2:** esse tutorial vai considerar a implementação da instrução implementada por esse grupo, o de MAIOR (maior rx, ry, rz - analisa o maior entre ry e rz e armazena em rx). Assim, será levado em consideração a instrução que implementamos. Como a nossa instrução utiliza três registradores, acabamos por seguir o padrão de código já usado em instruções semelhantes, como ADD, SUB, MUL, … (em algumas partes, basicamente, é um copia e cola). Portanto, recomenda-se seguir a mesma lógica, caso for criar instruções que utilizam, por exemplo, 2 registradores.

- Após a implementação das instruções padrões (aquelas que o professor completou em sala) e da escolha da nova instrução, vão ser necessários as alterações nos arquivos seguintes:
  - **defs.h** (está na pasta montador_fonte): é nesse arquivo que conterá os códigos das instruções. Primeiro, será necessário definir o opcode "interno" (deve ser um número que ainda não foi utilizado e menor que 100).

```
105 /* a nova instrução */
106 #define MAIOR_CODE                98
```

    Em segundo lugar, precisamos definir o binários da função. Como fizemos uma instrução aritmética, o nosso deve começar com 10, nos lógicos devem começar com 01.

```
135 /* a nova instrução */
136 #define MAIOR           "100110"
```

    Em terceiro lugar, precisamos definir o nome da instrução.

```
320 /* MAIOR */
321 #define MAIOR_STR                    "MAIOR"
```

- **montador.c** (está na pasta montador_fonte): nessa etapa, por nossa instrução ser, praticamente, iguais as de 3 registradores, seguimos o padrão das outras

```
198             /* Instrucoes de 3 argumentos e 1 linha : instr (), (), () -> [...] */
199             case ADD_CODE :
200             case ADDC_CODE :
201             case SUB_CODE :
202             case SUBC_CODE :
203             case MUL_CODE :
204             case DIV_CODE :
205             case LMOD_CODE :
206             case AND_CODE :
207             case OR_CODE :
208             case XOR_CODE :
209             case MAIOR_CODE :
210                 parser_SkipUntil(',');
211                 parser_SkipUntil(',');
212                 parser_SkipUntilEnd();
213                 end_cnt++;
214                 break;
```

```
778                 /* ==============
779                    MAIOR Rx, Ry, Rz
780                    ==============
781                 */
782
783                 case MAIOR_CODE :
784                     str_tmp1 = parser_GetItem_s();
785                     val1 = BuscaRegistrador(str_tmp1);
786                     free(str_tmp1);
787                     parser_Match(',');
788                     str_tmp2 = parser_GetItem_s();
789                     val2 = BuscaRegistrador(str_tmp2);
790                     free(str_tmp2);
791                     parser_Match(',');
792                     str_tmp3 = parser_GetItem_s();
793                     val3 = BuscaRegistrador(str_tmp3);
794                     free(str_tmp3);
795                     str_tmp1 = ConverteRegistrador(val1);
796                     str_tmp2 = ConverteRegistrador(val2);
797                     str_tmp3 = ConverteRegistrador(val3);
798                     sprintf(str_msg,"%s%s%s%s0",MAIOR,str_tmp1,str_tmp2,str_tmp3);
799                     free(str_tmp1);
800                     free(str_tmp2);
801                     free(str_tmp3);
802                     parser_Write_Inst(str_msg,end_cnt);
803                     end_cnt += 1;
804                     break;
```

```
2275    else if (strcmp(str_tmp,MUL_STR) == 0)
2276    {
2277        return MUL_CODE;
2278    }
2279    else if (strcmp(str_tmp,MAIOR_STR) == 0)
2280    {
2281        return MAIOR_CODE;
2282    }
```

- Feitos essas modificações é necessário ir para o terminal. Vá para a pasta que contém esses arquivos e rode o seguinte código:

  | gcc *.c -o montador |

  Após executar esse código será gerado um montador

- **simple_simulator_template:** é nesse arquivo que será implementada a lógica por trás da instrução (lembrando que essas modificações foram feitas para a nossa instrução, no seu pode mudar a forma de implementar). Primeiro, pegue o código do binário do defs.h que foi definido e transforme para decimal e faça:

```
78 // Opcode da nova instrução
79 #define MAIOR 38 //"100110"
```

Em segundo lugar, no case STATE_DECODE, foi alterado (copia e cola dos case dos aritméticos):

```
600                                 case MAIOR:
601                                     selM3 = ry;
602                                     selM4 = rz;
603                                     selM2 = sULA;
604                                     OP = opcode;
605                                     carry = 0;
606                                     LoadReg[rx] = 1;
607                                     selM6 = sULA;
608                                     LoadFR = 1;
609                                     // ----------------
610                                     state=STATE_FETCH;
611                                     break;
```

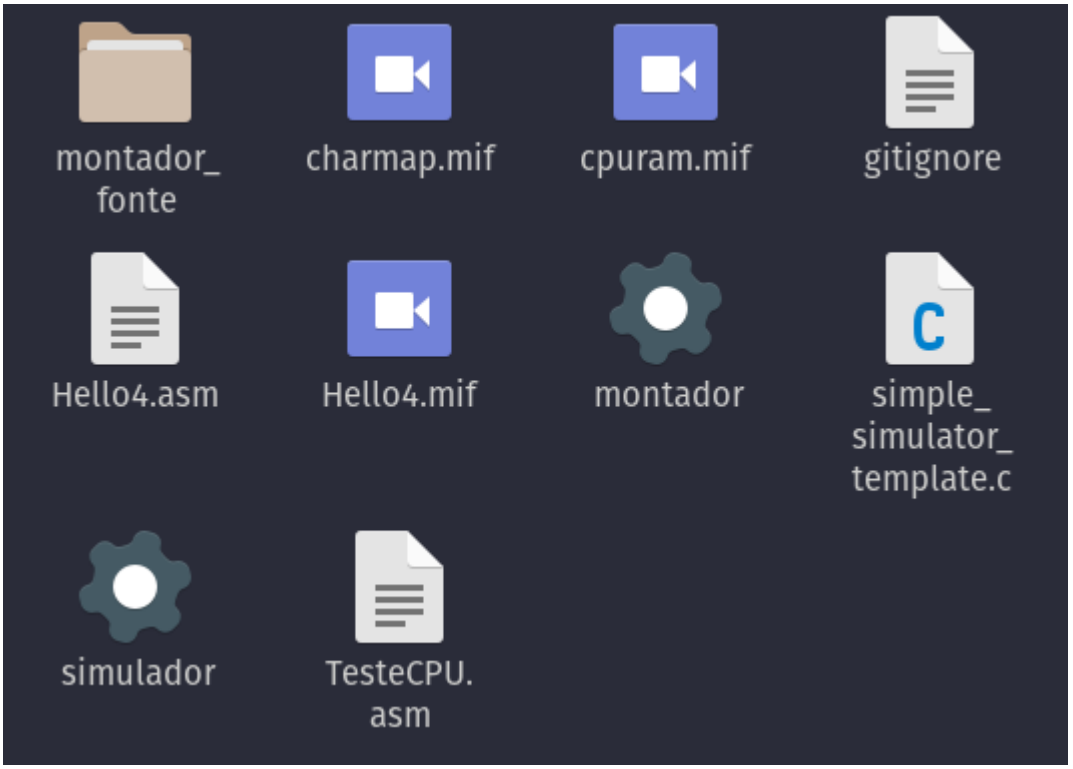
Em terceiro lugar, na função ResultadoUla modifique, de acordo com a lógica da sua instrução:

```
925                                 case MAIOR:
926                                         if (x > y){
927                                                 result = x;
928                                         } else {
929                                                 result = y;
930                                         }
931
932                                         break;
```

- **TesteCPU.asm:** modifique esse arquivo para conseguir testar a instrução de acordo com que irá implementar

```
; Teste do Maior
loadn r0, #2
loadn r1, #3
maior r2, r1, r0
cmp r2, r1
jne Fim
loadn r0, #'Z'
loadn r1, #32
outchar r0, r1
jmp Fim
```

- Feito essas modificações, vá para o terminal com todos os esses arquivos em uma mesma pasta (o montador gerado anteriormente deve estar junto desses arquivos)



- Rode esse código para montar a cpuram.mif

  | ./montador TesteCPU.asm cpuram.mif |

- Rode esse código para gerar o simulador

  | gcc simple_simulator_template.c -O3 -march=native -o simulador -Wall -lm |

- Rode esse código para rodar os testes

  | ./simulador |

- Esse foi o resultado esperado para a nossa instrução:

```
└─(01:37:00)──> ./simulador
PROCESSADOR ICMC  - Menu:
                          'r' goto inicio...
                          'q' goto fim...

Rodando...
ABCDEFGHIJKLMNOPZ
```

✨Parabéns por concluir o processador!!!✨