```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

uint32_t Instruction_Memory[0x2004];
uint32_t Data_Memory[0x4000];
uint32_t PC_Next=0;
uint32_t Register_File[32];
uint32_t PCPlus4=0;
uint8_t RegWrite = 0;
uint8_t ImmSrc = 0;
uint8_t ALUSrc = 0;
uint8_t MemWrite = 0;
uint8_t ResultSrc = 0;
uint8_t Branch = 0;
uint8_t ALUOp = 0;
uint32_t ALUcontrol = 0;


////////////////////////////////////////////////////////////////
uint32_t Get_Instruction(uint32_t address){ //ინსტრუქციის ამოღება ინსტრუქციონ მეხსიერიდან
    uint32_t temp = address/4;

    return Instruction_Memory[temp];
}

void printregister(void){ // რეგისტრ ფაილის რეგისტრების გამოტანა ტერმინალში
    for(int i=0; i <4; i++ ){
        for(int j =0; j <8; j++ ){
                    printf("r%d: %x \t", i * 8 + j,Register_File[i * 8 + j]);
        }
        printf("\n");
    }




}

uint32_t get_bits(uint32_t Instruction, uint32_t start_bit, uint32_t lenght)
{ //ინსტრუქციიდან ბიტების ამოღება საწყისი ბიტის და სიგრძის მიხედვით
   uint32_t r = (Instruction >> (start_bit));
   uint32_t mask = ((1 << lenght)-1); //ამოღება ხდება მასკით

   return mask & r;
}

uint32_t extender(uint32_t _instruction, uint8_t _ImmSrc){ //იმედიეითის ექსტენდერი
    uint32_t imm=0;
    uint32_t tempImm = 0;

   switch(_ImmSrc){// სხვადასხვა ტიპის ინსტრუქციაში იცვლება იმედიეითის მდებარეობა
            case 0: // I ტიპისთვის
                imm = get_bits(_instruction, 20, 12);
                break;
            case 1: // S ტიპისთვის
                tempImm = get_bits(_instruction, 7, 5);
                imm = get_bits(_instruction, 25, 7);
                imm = imm << 5;
                imm = imm | tempImm;
                break;
            default:
                return 0;
        }



    uint32_t mask1 = 0b00000000000000000000100000000000; // უარყოფითი/დადებიდი ნიშნის ამსაღები მასკი
    uint32_t sign = imm & mask1;

    if (sign != 0){
        uint32_t mask2 = 0b11111111111111111111000000000000; //mask
        return imm  |  mask2;
    }
    return imm;

}

uint32_t ALU(uint32_t ScrA, uint32_t ScrB, uint8_t ALUControl){

    switch(ALUControl){
    case 0b000:
        return ScrA + ScrB;
```

```
84      case 0b011:
85          return ScrA | ScrB;
86
87      default:
88          return 0;
89      }
90
91  }
92  void Main_Decoder (uint32_t _Instruction){
93      uint32_t op = get_bits(_Instruction,0,7);
94      switch(op){
95      case 0b0000011:
96          RegWrite = 1;
97          ImmSrc = 0;
98          ALUSrc = 1;
99          MemWrite = 0;
100         ResultSrc = 1;
101         Branch = 0;
102         ALUOp = 0;
103         break;
104     case 0b0100011:
105         RegWrite = 0;
106         ImmSrc = 1;
107         ALUSrc = 1;
108         MemWrite = 1;
109         ResultSrc = 1;
110         Branch = 0;
111         ALUOp = 0;
112         break;
113     case 0b0110011:
114         RegWrite = 1;
115         ImmSrc = 0;
116         ALUSrc = 0;
117         MemWrite = 0;
118         ResultSrc = 0;
119         Branch = 0;
120         ALUOp = 2;
121         break;
122     default:
123         break;
124         }
125         printf("++++++++++++++++++++++++++++++++++ \n");
126         printf("RegWrite = %x\n\r", RegWrite );
127         printf("ImmSrc = %x\n\r", ImmSrc);
128         printf("ALUSrc  = %x\n\r", ALUSrc );
129         printf("MemWrite = %x\n\r", MemWrite);
130         printf("ResultSrc = %x\n\r", ResultSrc);
131         printf("Branch = %x\n\r", Branch);
132         printf("ALUOp = %x\n\r", ALUOp);
133         printf("++++++++++++++++++++++++++++++++++ \n");
134
135 }
136
137 void ALU_Decoder(uint32_t _Instruction){
138     uint32_t op_5 = get_bits(_Instruction,5,1);
139     uint32_t funct_3 = get_bits(_Instruction,12,3);
140     uint32_t funct7_5 = get_bits(_Instruction,25+4,1);
141     uint32_t op_5_funct7_5 = (op_5 <<1) + funct7_5;
142     switch(ALUOp){
143     case 0b00:
144       ALUcontrol= 0b000;
145       break;
146
147     case 0b01:
148       ALUcontrol= 0b001;
149
150        break;
151     case 0b10:
152         switch(funct_3){
153             case 0b000:
154                 switch(op_5_funct7_5 ){
155
156                         case 0b00:
157                         case 0b01:
158                         case 0b10:
159                             ALUcontrol= 0b000;//(add)
160                             break;
161                         case 0b11:
162                             ALUcontrol= 0b001;//(subtract)
163                             break;
164                         default:
165                             break;
166                     }
167                 break;
```

```
168                 case 0b010:
169                     ALUcontrol= 0b101;//(set less than)
170                     break;
171                 case 0b110:
172                      ALUcontrol= 0b011;//(or)
173                     break;
174                 case 0b111:
175                      ALUcontrol= 0b010;//(and)
176                     break;
177                 default:
178                     break;
179 
180             }
181 
182 
183           break;
184         default:
185             break;
186         }
187 
188 
189     }
190 
191 
192 
193 
194 
195     int main()
196     {
197         //წინა ბრძანება
198         PC_Next=0x1000;
199 
200         Instruction_Memory[0x1000 / 4] = 0xFFC4A303;
201         Instruction_Memory[0x1004 / 4] = 0x0064A423;
202         Instruction_Memory[0x1008 / 4] = 0x0062E233;
203         Data_Memory[0x2000] = 10;
204         Register_File[5] = 6;
205         Register_File[9] = 0x2004;
206 
207         for(int x =0; x <3; x++ )
208         {
209 
210 
211 
212                 uint32_t Instruction = Get_Instruction(PC_Next);
213                 printf("Instruction = %x\n\r", Instruction);
214 
215                 printregister();
216 
217                 Main_Decoder(Instruction);
218                 ALU_Decoder(Instruction);
219 
220                 uint32_t A1 = get_bits(Instruction,15,5);
221                 printf("A1 = %x\n\r", A1);
222 
223                 uint32_t A2 = get_bits(Instruction,20,5);
224                 printf("A2 = %x\n\r", A2);
225 
226                 uint32_t A3 = get_bits(Instruction,7,5);
227                 printf("A3 = %x\n\r", A3);
228 
229                 uint32_t RD1= Register_File[A1];
230                 printf("RD1 = %x\n\r", RD1);
231 
232                 uint32_t RD2= Register_File[A2];
233                 printf("RD2 = %x\n\r", RD2);
234 
235                 uint32_t immExt = extender(Instruction,ImmSrc);
236                 printf("immExt - %x\n\r", immExt);
237 
238 
239 
240                 uint32_t SrcB = 0;
241                 if(ALUSrc==1)
242                 {
243                     SrcB = immExt;
244                 }
245                 else
246                 {
247                     SrcB = RD2;
248                 }
249 
250                 uint32_t  SrcA = RD1;
251                 uint32_t  ALUResult = ALU(SrcA,SrcB,ALUcontrol);
```

```
252
253                 printf("SrcA - %x\n\r ",SrcA);
254                 printf("SrcB - %x\n\r ",SrcB);
255                 printf("ALUResult - %x\n\r ",ALUResult);
256
257
258
259
260
261                 uint32_t ReadData = Data_Memory[ALUResult];
262
263                 uint32_t WD3=0;
264                 if(ResultSrc==1)
265                 {
266                     WD3 = ReadData;
267                 }
268                 else
269                 {
270                     WD3 = ALUResult;
271                 }
272
273                 printf("WD3 - %x\n\r ",WD3);
274
275                 //#####################################################
276                 //#####################################################
277                 //#####################################################
278                 //       ak moxda clockis impulsis mosvla    !!!!!!!!!!!!!!!!
279               //#####################################################
280                 //#####################################################
281                 //#####################################################
282
283                 if (MemWrite == 1)
284                 {
285                    Data_Memory[ALUResult]= RD2;
286                 }
287
288                 if (RegWrite == 1) // WE3 = RegWrite
289                 {
290                    Register_File[A3]= WD3;
291                 }
292
293                 PCPlus4 = PC_Next;
294                 PCPlus4 = PCPlus4+4;
295                 PC_Next = PCPlus4;
296                 printf("PCNext - %x\n\r ",PC_Next);
297
298                 printf("------------------------------------------------------- \n\n\n");
299
300       }
301
302                 //printregister();
303
304       return 0;
305   }
306
```