INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Engenharia de Eletrónica e Telecomunicações e de Computadores

Engenharia Informática e de Computadores

4.º Trabalho Prático

de

Arquitetura de Computadores

Interação com dispositivos externos
(Jogo da roleta)

G2:
44873 – Paulo Rosa
44808 – Ricardo Pinto

Docente: Prof. Tiago Silva

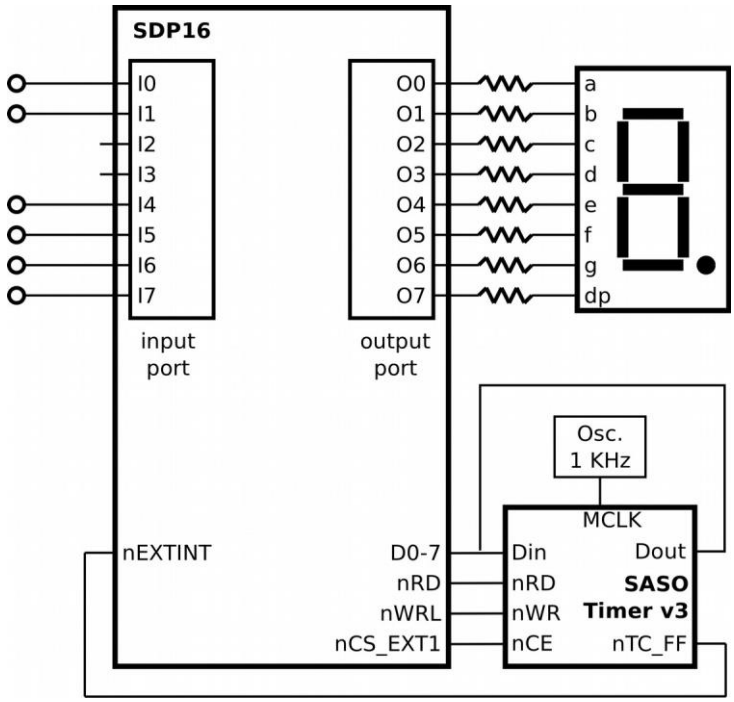
2 de dezembro de 2019

1 Objetivos

O presente trabalho dá aos alunos oportunidade para usar e melhor compreender o *hardware* envolvente de um microprocessador e para treinar a programação em linguagem *assembly*, tirando partido do ambiente de desenvolvimento baseado no sistema didático SDP16. Estão envolvidos os seguintes tópicos: entrada e saída de dados; interrupções; temporização; organização de código e estruturação em rotinas.

2 Descrição do trabalho

Pretende-se implementar uma versão simplificada do jogo da roleta, recorrendo ao sistema didático SDP16, ao temporizador SASO_Timer_v3 (em anexo) e a um mostrador de 7 segmentos, conforme ilustrado na figura.



Nesta versão do jogo da roleta, a roda contém apenas 16 números, de 0 a 15, e é simulada pelo mostrador de 7 segmentos. O interruptor I0 é utilizado para pôr a roda a girar, enquanto o botão I1 serve para bloquear a aposta do jogador. Os interruptores I4 a I7 são utilizados para definir o número em que o jogador aposta. O temporizador SASO_Timer_v3 é utilizado para realizar a base de tempo necessária ao funcionamento da aplicação.

Relativamente à sequência do jogo, cada jogada desenrola-se em cinco fases:

1. A jogada inicia-se com a ativação do interruptor I0, que coloca a roda a girar. Este movimento é simulado pela iluminação sucessiva de um segmento do mostrador de 7 segmentos, aparentando um movimento circular.
2. Enquanto a roda está a girar, é permitido que o jogador defina e altere livremente o valor da sua aposta, por manipulação dos interruptores I4 a I7.

3. A ativação do interruptor I1 serve para confirmar a aposta e faz acender o ponto do mostrador de 7 segmentos para sinalizar este evento.

4. Com a confirmação do valor da aposta, fica bloqueada a possibilidade de alteração da aposta mas a roda irá girar ainda durante mais algum tempo, um valor aleatório, no intervalo 5 s a 10 s.

5. Quando a roda parar de girar, o número sorteado, que deve ser gerado de forma aleatória, é afixado no mostrador usando o código hexadecimal.

A geração de números aleatórios pode basear-se no valor do contador do temporizador SASO_Timer_v3, atendendo a que este periférico está oculto do utilizador e, portanto, o jogador não tem forma de relacionar o momento em que é obtida a semente para o gerador de números aleatórios com o valor atual do contador.

3 Execução faseada

O presente trabalho prático será realizado em duas fases. Na primeira fase, os alunos deverão instalar o temporizador SASO_Timer_v3 e construir um pequeno programa de teste que verifique o acesso aos portos do sistema didático SDP16 e a utilização do temporizador, ora por pesquisa de estado (recorrendo ao porto de entrada), ora por atendimento de interrupção. Na segunda fase, os alunos realizarão o sistema proposto neste enunciado.

4 Questões para serem respondidas no relatório

1. Explique os cálculos que realizou para determinar as temporizações envolvidas neste trabalho.
2. Qual a latência máxima do sistema no atendimento do temporizador?
3. No pior caso, quanto tempo demora a execução da rotina utilizada para o atendimento da interrupção?

5 Avaliação

O trabalho é realizado em grupo, conta para o processo de avaliação da unidade curricular, estando sujeito a discussão final, e tem a duração de três semanas.

A apresentação das soluções propostas por cada grupo para cada uma das fases do trabalho decorre em sessão de laboratório, em data a combinar com o docente responsável pela lecionação das aulas da respetiva turma.

Após esta apresentação, cada grupo deverá entregar o relatório do trabalho ao docente, no qual deve constar:

- Descrição dos elementos relevantes para a compreensão do trabalho realizado;
- Resposta às perguntas formuladas no enunciado;
- Conclusões;
- Listagens dos programas realizados (.lst)

Primeira Fase

"Os alunos deverão instalar o temporizador SASO_Timer_v3 e construir um pequeno programa de teste que verifique o acesso aos portos do sistema didático SDP16 e a utilização do temporizador, ora por pesquisa de estado (recorrendo ao porto de entrada), ora por atendimento de interrupção"

1 – Acesso aos portos (escrita e leitura), sem interrupção.

Montagem:

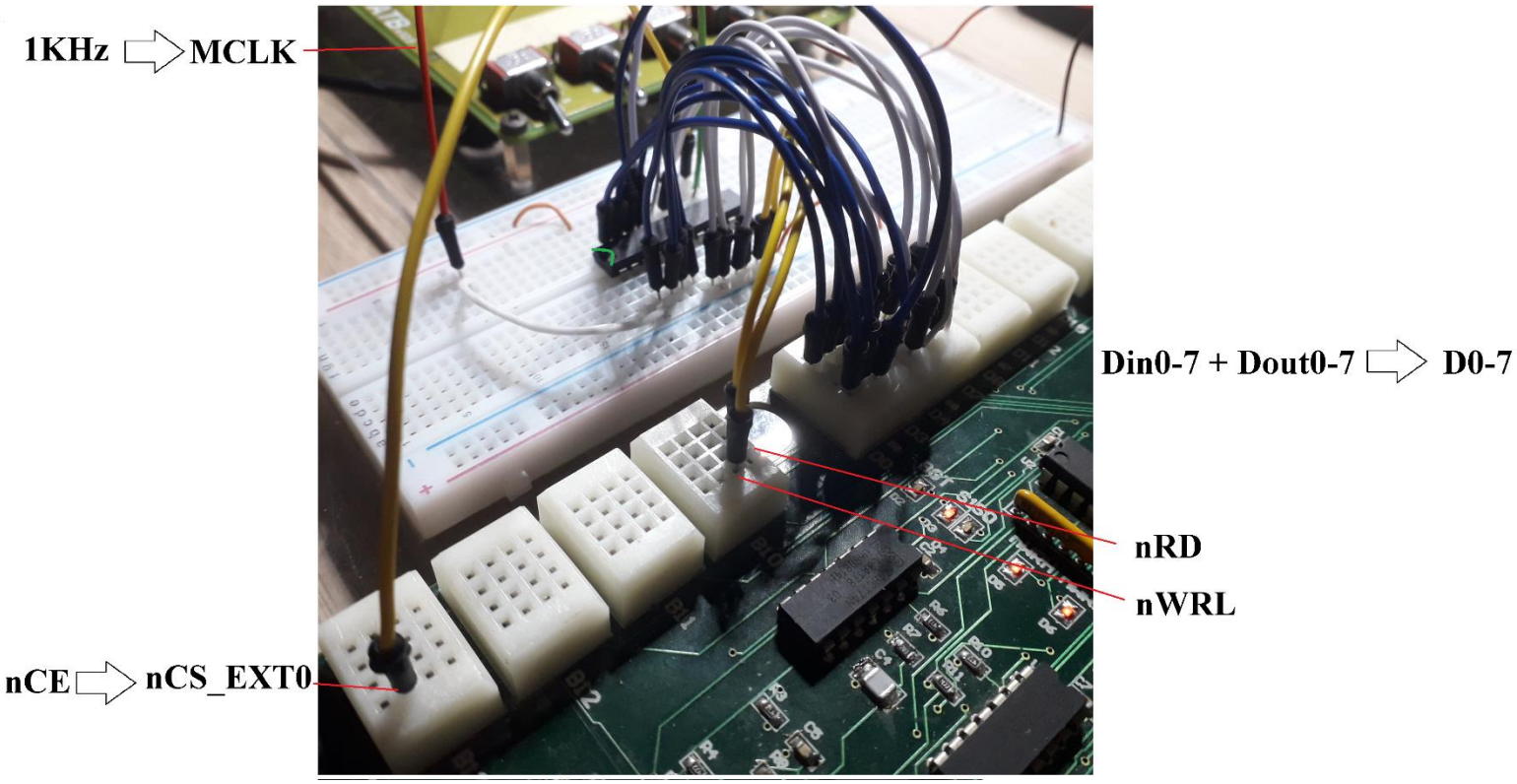


Figura 1, ligação com o timer fase 1

Resolução:

A gama de endereços que foi escolhida para aceder ao SASO TIMER é de 0xFF40 até 0xFF7F(e é por essa razão, segundo o "Mapa de memória e descrição em CUPL do gerador de CS (PAL U12)", que ligámos o nCE (do SASO TIMER) ao nCS_EXT0 (do P16).

Em primeiro lugar, definimos o símbolo SASO_ADDR com o valor: 0xFF40

Criámos a label "timer_init" que escreve um valor dado como input(r0) no LR do timer. É carregado para r1 o valor que está associado ao símbolo SASO_ADDR, fazemos store byte do valor que está em r0 nessa posição de memória e voltamos à posição a seguir da chamada desta função (fazendo "mov pc,lr").

Criámos a label "getvalue", em que a única diferença da anterior é que em vez de se fazer store byte, faz-se load byte. Portanto, esta função lê o valor que está no timer no momento.

Também criámos a label “loopgetvalue”,(em que se assume que/faz-se com que em r1 já esteja o endereço do timer) para ser um bocado mais rápido(menos uma linha) ler o valor presente no timer, ao executar estas linhas no debugger (ao metermos o MCLK ligado a 1Hz), podemos ver passo a passo o decremento um a um da contagem do timer. É uma forma fácil de confirmar que tudo está a funcionar corretamente.

2 - Acesso aos portos (escrita e leitura, com interrupção).

Montagem: É acrescentada à montagem anterior a ligação da saída nTC_FF do timer à entrada nEXTINT do PDS16. Assim, sempre que o CR do timer chega a #0, é assinalado ao P16 um pedido de interrupção.

Resolução: É acrescentado ao código anterior a label “enableisr”(que vem depois na iniciação do valor do timer). Esta função passa o valor do CPSR (current program state registers) para o r12. Passamos para o SPSR (saved program state registers) o valor que foi guardado em r12 (o CPSR), Passamos a flag IE (interrupt enable) para o valor lógico 1. Logo, assim que a entrada nEXTINT está a 0, o PC automaticamente vai para a posição 0x0002, e é feito *branch* para a label “isr” que só faz a leitura do valor que está no timer no instante.

Segunda Fase

Para a segunda fase, acrescenta-se ao projeto o mostrador de 7-segmentos. Usámos o nosso mostrado (Fig 1), de dimensão média, que é do tipo *common cathode* (os pins do meio de cima e de baixo levam *ground*) e os restantes são para controlar os LEDs (e devem-se usar resistências para os pins dos LEDs).

Esta (Fig 2) é a configuração que associa os bits com a localização dos LEDs que escolhemos e que fizemos tendo em conta que vamos ter de fazer uma animação rotativa. A animação corre, por ordem, os LEDs A, B, C, D, E, F. Logo, os bits de 5-0 serão os LEDs de F-A, porque assim só precisamos de um bit (que começa do A) e fazemos logical shift left desse bit até ao bit 5 (e repete-se do A). E assim configuramos a animação rotativa.

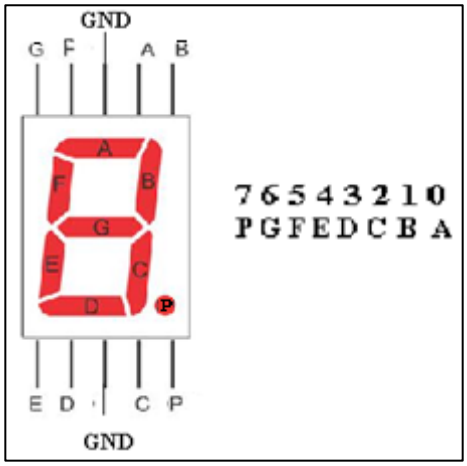


Fig 2

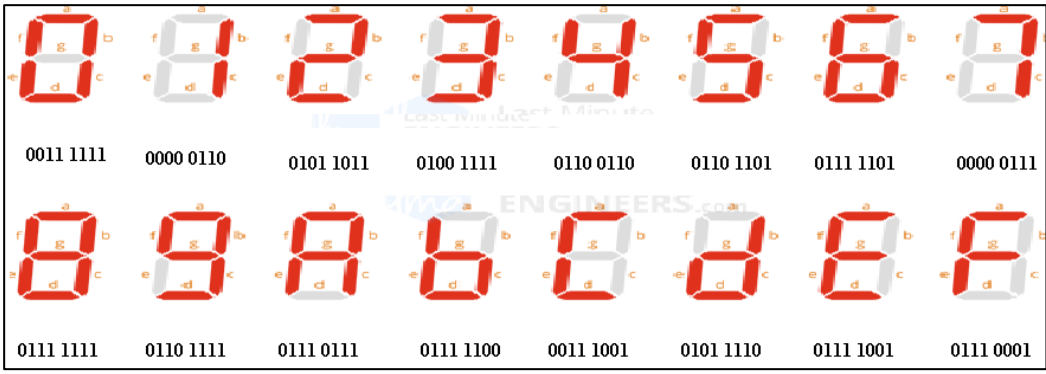


Fig 3

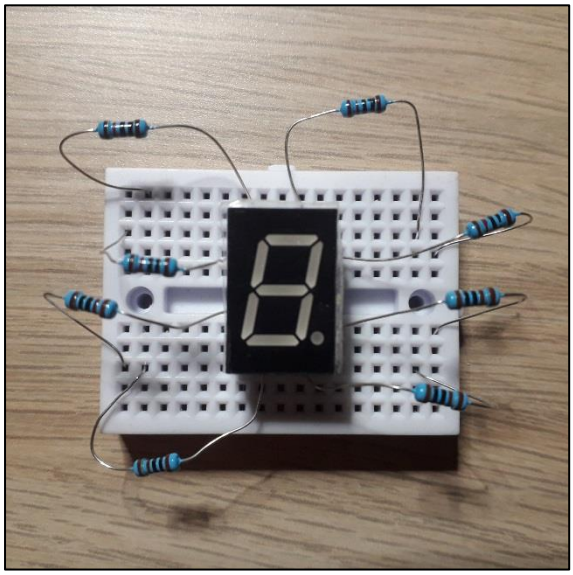


Figura 4

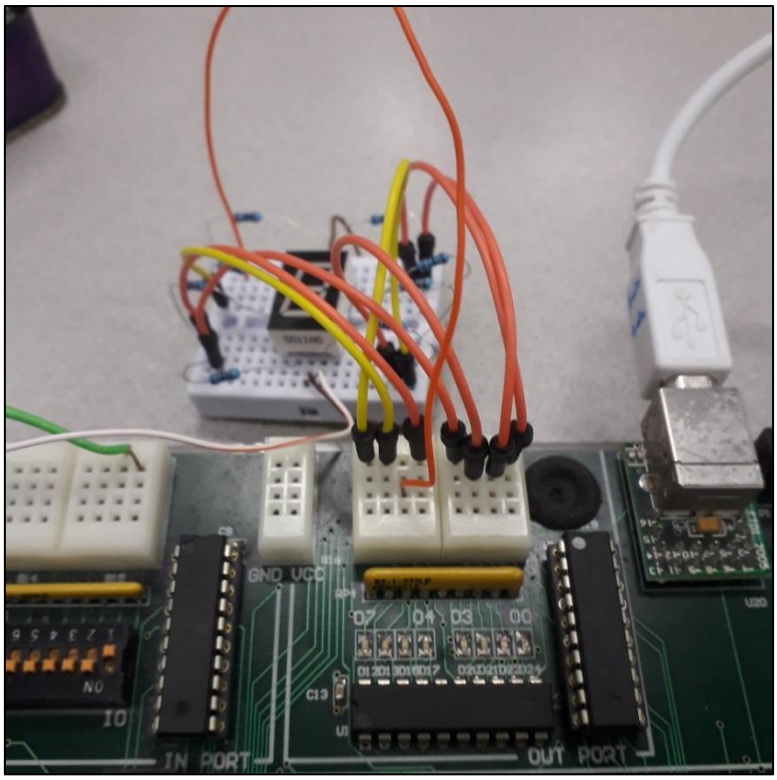


Figura 5

Para esta fase, acedemos ao I/O Port, que é acessível do endereço 0xFF00 ATÉ 0xFF3F. Então, definimos o símbolo: SDP16_PORTS_ADDRESS com o valor associado: 0xFF00.

Estado 0 –

É iniciado o SASO timer com 0b11111111. Portanto, este faz a contagem decrescente a partir deste valor. Depois, passamos logo para o estado seguinte.

Estado 1 –

Para o estado inicial deste jogo, definimos que o ponto do 7-segment, fica a piscar com a mesma frequência que o bit mais significativo que é lido do SASO TIMER. Este acende e apaga de 128 em 128 ticks. (1000 0000). E a frequência do MCLK é 1Khz, logo 128*0,001s (1khz) = 0.128s. Apaga e acende a uma velocidade de 0.128s.

E permanece a piscar até o botão I0 ser pressionado. Que avaliamos ao fazer *branch link* para "button0", que faz a leitura (load) so port_input. Fazemos mask to byte menos significativo e, se esse valor não for 1, continuamos neste estado. Se este bit estiver a 1, avança-se para o estado 2.

Estado 2 –

Neste estado ocorre a animação giratória da roleta. É chamada a função "spin" que recebe um input que basta ser do tipo byte. Neste caso, pretendemos que o input seja #0. Assim, é realizada a animação da roleta e sem ponto (porque ainda não se carregou no I1 para confirmar a aposta).

Portanto, neste estado, temos um ciclo principal, que se chama "spin"(que é usado no início ou no fim de um ciclo completo) que inicia o r5 (o registo que contém o único bit que corresponde ao LED que é para acender) a 1# e r4 a #64 (que corresponde ao último LED que é aceso do ciclo) No início do "cycle", fazemos *clear* (write com #0) do port_output (mas antes de fazer isso, fazemos uma operação "or" com o #0 e o valor dado como *input* (e segundo este valor, é acrescentado o bit que acende o ponto ou não)

Depois, chamo o método "delay", que faz a subtração de um valor (até este chegar a 0) que defini como sendo o símbolo TIME_DELAY, com o valor 250. Portanto, é o suficiente para gerar um intervalo de tempo entre os vários LEDs acenderem e apagarem para formar uma boa animação.

A seguir, meto no r0 o valor que é produzido entre o *input* que é dado a esta função (o bit para suportar a diferença entre ser a animação giratória com ponto, como acontece no próximo estado) e o valor do LED que é acender neste ciclo (que está no r5). Depois, é feito *shift left* do r5, para acender LED posterior deste (em relação ao sentido da animação rotativa), que é para acender no próximo passo.

Depois é feita a confirmação se este ciclo é com ponto ou não (porque se for sem ponto, queremos verificar se I1 é pressionado), mas no próximo estado (estado 3, o do ponto), não queremos verificar nada relacionado com a ativação do botão I1 (portanto salta-se esta verificação) usando a *label* "componto".

Logo, como neste estado o *input* é #0, vai verificar-se a ativação do I1, e se não estiver ativado, continua-se o ciclo normalmente. Se estiver ativado, continua-se a animação rotativa dos LEDs tal como estavam mas, retorna-se ao ciclo com o *input* #1, para ativar o ponto.

No fim desta função, é comparado o valor do LEDs corrente que foi acendido(r5) e o r4, valor fixo, igual a #64. Se os valores forem iguais, fazemos *jump* para "spin" para recomeçar o ciclo da animação.

E só se avança para o próximo estado quando I1 for pressionado e nesse instante armazena-se o valor que se apostou, em memória, que se encontra de I7 a I4 no input-port. E é gerado (lendo o valor corrente do timer) um número aleatório que é também guardado em memória. Decidimos gerar aqui, porque ao tentarmos gerar o número aleatório no fim da contagem do tempo aleatório entre 5s e 10s, os valores não saíram tão aleatórios como seria previsto.

O que aconteceu, é que, ao fazermos a leitura do valor presente no timer, estávamos à espera de um valor aleatório, mas esta leitura ocorre sincronizada com a ocorrência da interrupção. Isto significa que o CR acabou de passar por 0, que é quanto atua "nTC_FF" (que provoca interrupção). Pelo que recebemos valores muito repetidos.

Estado 3 –

Neste estado, é continuada a animação giratória tal como estava, mas desta vez o ponto do 7-segment está ligado. É ignorada a verificação da ativação do botão I1 e é ativada a aceitação de interrupções. E ocorre a contagem de um intervalo aleatório entre 5 e 10 segundos.

Para realizar a contagem aleatória, começamos com 5s, a estes 5s, vai ser adicionado um valor aleatório entre 0 e 5s. E o valor destes 5s é decrementado (1 a 1) até que a sua soma com o valor aleatório seja inferior a 0. Portanto 5s, temos a certeza que passam, porque quando este valor, que é decrementado sucessivamente, chegar a 0, passaram 5s, e ainda é feita a soma com um valor entre 0 e 5. Logo, dá um valor igual ou superior a 0. E no caso em que ocorre mais tempo, será quando o valor que é decrementado chegar a -20 (ticks de "nTC_FF"), e o valor aleatório for 20, nesse caso passam 10s.

Para o cálculo do tempo aleatório entre 5-10:. sabemos que a animação rotativa com ponto deve manter-se ativa durante pelo menos 5s. O MCLK está a 1Khz, que é igual a 0.001s. O pedido de interrupção está por conta do valor "nTC_FF". Este valor fica a 0 (e que está ligado a "nEXTINT"), quando o CR chega 0b00000000. Logo, de 0b11111111 a esta ocorrência, são 2^8-1(255) ticks. Fazendo os cálculos ocorre um pedido de interrupção a cada: 255 * 0.001 = 0.255s. Queremos contar 5s. Logo 5s/0.255 ~= 20 contagens (ticks de "nTC_FF"). Usamos o "nTC_FF" como clock para contabilizar o tempo.

Começamos com a função "enableisr", que guarda as *flags* no r12. Depois, alteramos o IE para o valor 1 do CPSR (current program status register), que automaticamente, faz com que o P16 permita pedidos de interrupção. E no SPSR (saved program status register) é guardado as flags e com o IE a 1. E o CPSR fica com M = 1 (modo de interrupção). Ao acontecer isto, o PC será igual a 0x0002.

Quando se chegar a este ponto, é feito um *jump* para a *label* "isr". Que primeiro vai buscar um valor aleatório ao *timer*, faz-se mask dos primeiros 4 bits menos significantes e confirma-se se o seu valor é igual ou inferior a 20(5s), para somar ao tempo restante que falta, um valor entre 0 e 5s. (0-20ticks de "nTC_FF").

Passado este tempo, segue-se para a *label* "endisr". Que faz *reset* do valor "waittime" para 20, para ser contabilizado os 5s no próximo jogo que ocorrer. E segue-se para a *label* "draw".

Estado 4 –

Na função "draw", temos um método que carrega o valor aleatório que saiu. É carregado o endereço do array com os vários valores, em byte, que definam a forma como é representada cada valor que é possível formar com 4 bits no formato hexadecimal no 7-segment. E é somado a este endereço o valor aleatório que saiu, o que automaticamente seleciona a forma/o número em hexadecimal que é para representar.

E é feita a comparação entre o número que saiu e o que foi escolhido. Se acertámos, segue-se para a função *label* "correct" que adiciona ao byte que foi lido da posição do array, um bit a 1 na posição mais significativa, de modo a acender o ponto do 7-segment, indicando que acertamos com o valor que apostamos. Se errámos, segue-se para a *label* "incorrect", mas que só faz *store* do byte no port_output, não ligando o ponto. E ambas fazem *branch* para a *label* "restart" que vai para a *label* "init" (estado 1) para recomeçar o jogo quando I1 for ativado.

I1 (o I1, é interpretado neste jogo como um botão de confirmação: confirmar aposta/confirmar o recomeço do jogo).

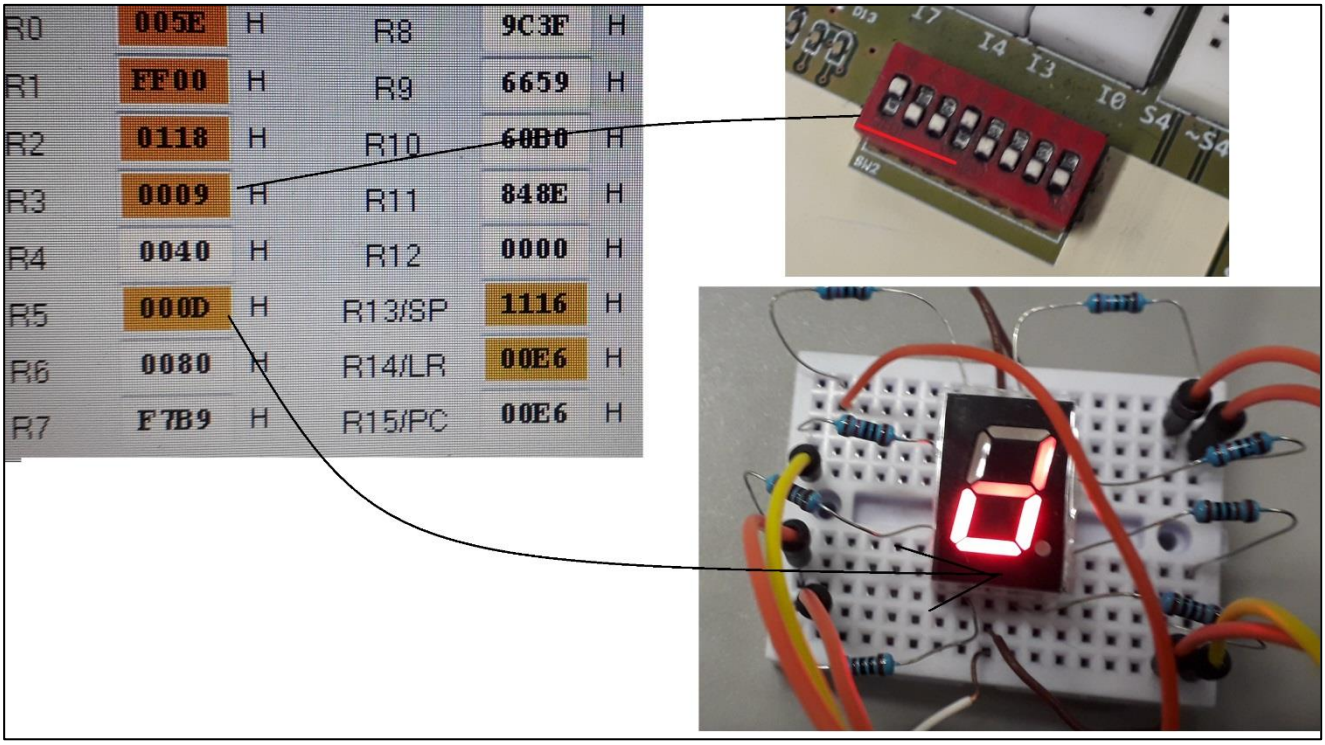


Figura 6, exemplo de como é apostado o valor 9, e saiu o valor D

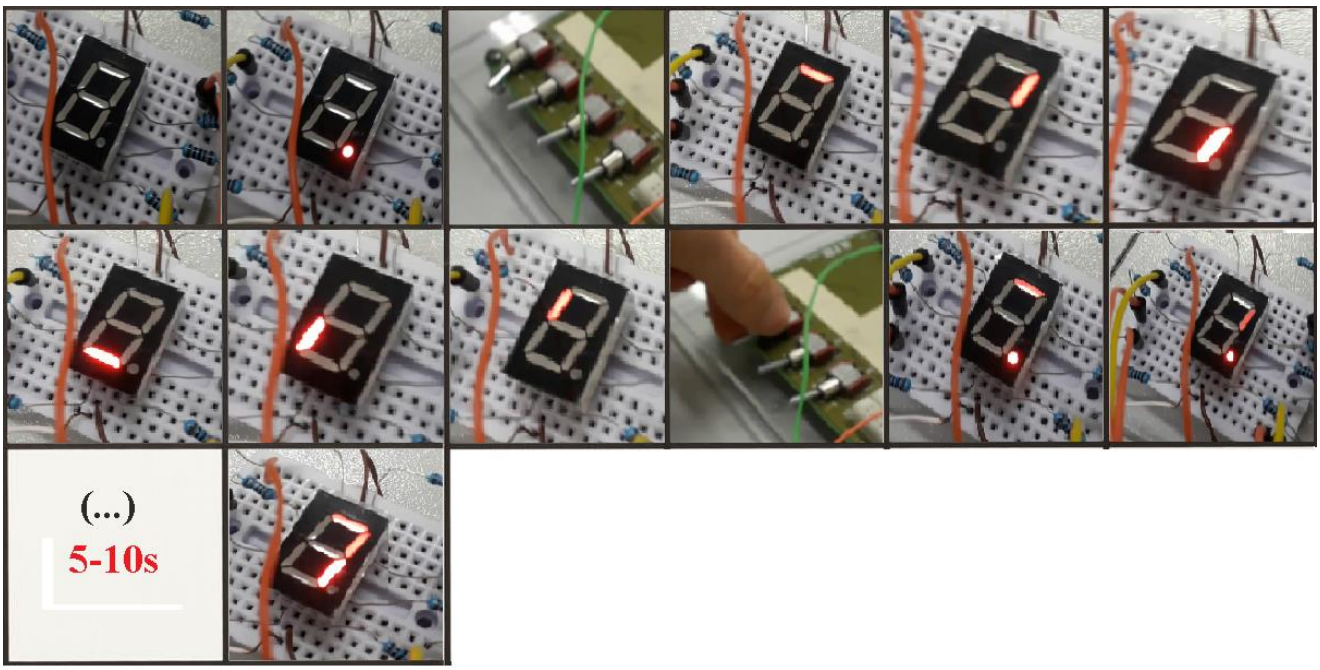


Figura 7, sequência de imagens a demonstrar o funcionamento do jogo.

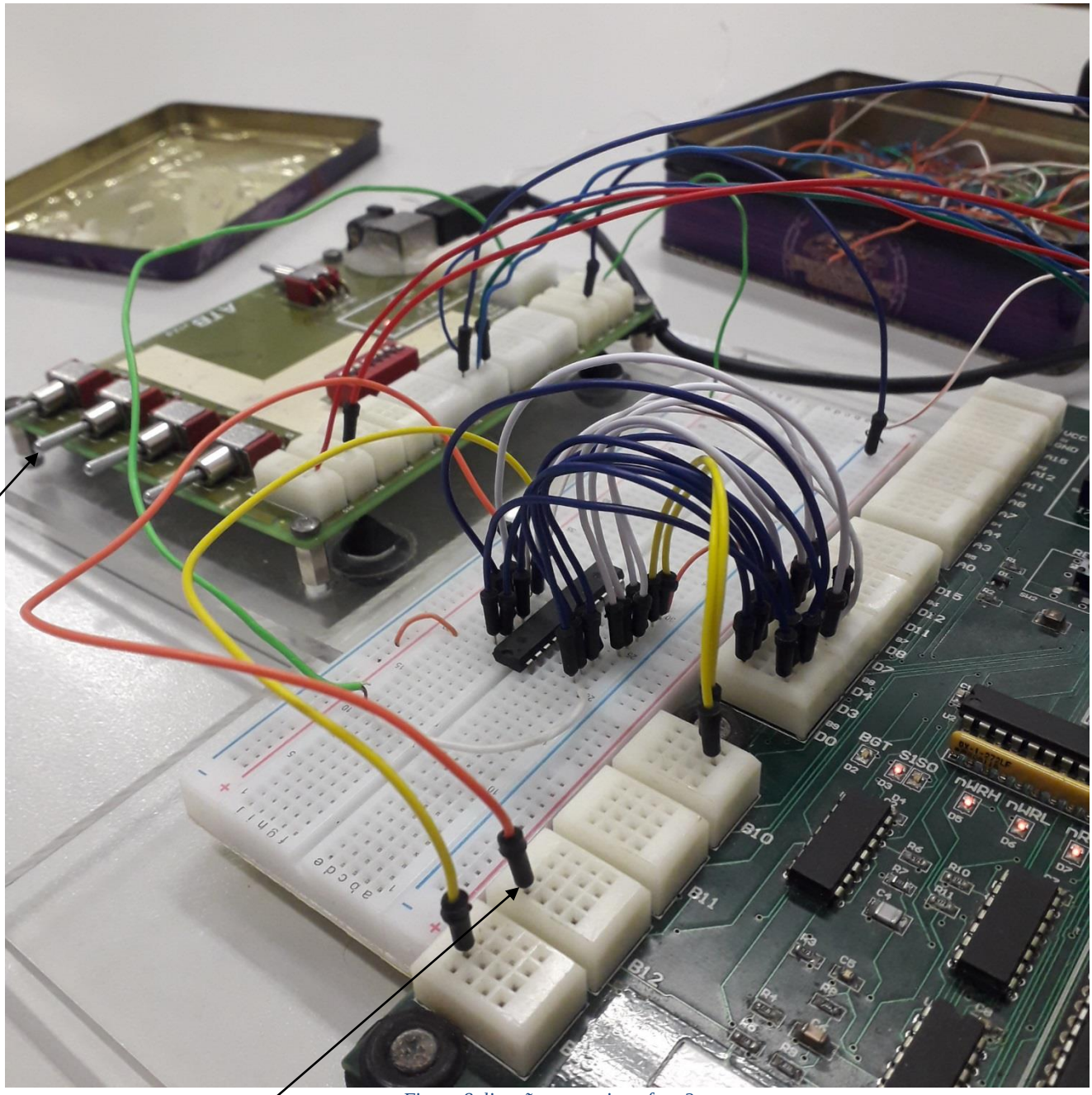


Figura 8, ligação com o timer fase 2

Resposta às perguntas:

1. Explique os cálculos que realizou para determinar as temporizações envolvidas neste trabalho.

O MCLK tem uma frequência de 1Khz. 1Khz = 0.001s.

O "nTC_FF" é ativado sempre que o timer chega #0, logo, passados 2^8-1(255) ticks.

Logo, tendo em conta a frequência do clock, ocorre "nTC_FF" a cada: 0.001s*255=0.255s

Logo, para contabilizar 5s. Faço 5s/0.255 ~= 20 (contagens/clocks de "nTC_FF")

2. Qual a latência máxima do sistema no atendimento do temporizador?

Sabe-se que o oscilador (MCLK) tem frequência: 500Hz. O tempo que passa para executar um ciclo máquina é, no máximo, 0.002s.

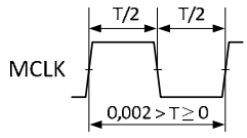


Figura 14-8 – *Master Clock*

Desde o momento em que o sinal "nTC_FF" fique a 0(e esse sinal faz com que nEXINT fique a 0), o o pior caso (tempo/latência mais alta) para que se chegue à rotina de interrupção é quando:

Se transita o IE no CPSR de 0 para 1, e o valor do timer no momento é de 0b00000001, portanto o "nTC_FF" está a 1. E para além disso, temos as instruções: "msr cpsr,r0" e "b isr".

Ciclos de relógio: 6 ciclos. 6*0.002 = 0.012s

Tempo: 0.012 + 0.255 = 0.267s

3. No pior caso, quanto tempo demora a execução da rotina utilizada para o atendimento da interrupção?

Todas as instruções demoram 3 ciclos de relógio para executar:

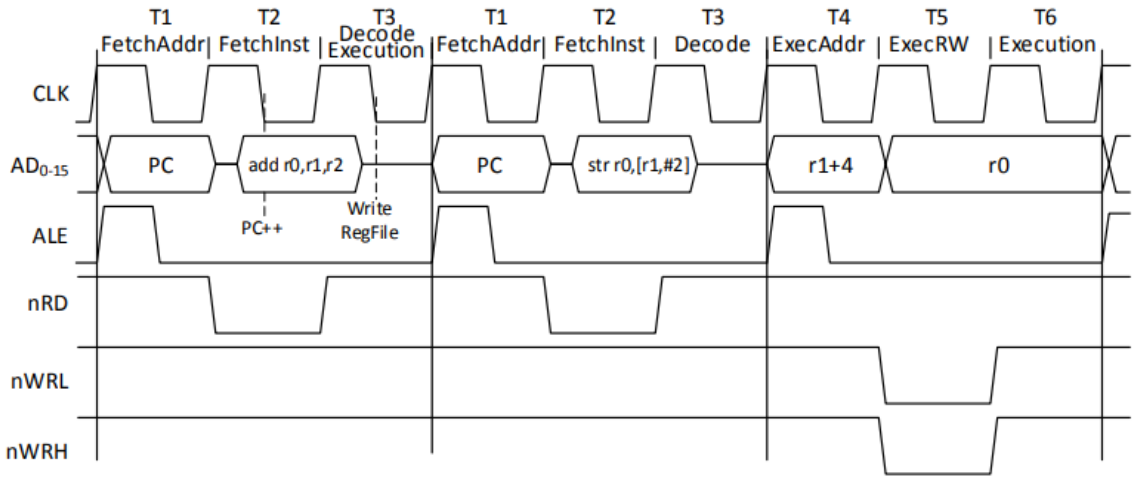


Figura 14-10 – Ciclos FETCH e EXECUTE

O ciclo OP CODE FETCH consiste numa sequência de 3 estados de T1 a T3, sendo T1 para preparação de endereço, T2 para leitura da instrução e T3 para descodificação e execução. No ciclo T3 caso se trate de uma instrução LDR ou STR, não existe execução e é precedido por T4, T5 e T6, ciclo de leitura caso a instrução em execução seja LDR, ou por um ciclo de escrita, caso a instrução seja STR. No caso da instrução STR, o estado de execução não realiza nenhuma acção internamente ao processador. No caso da instrução LDR, a meio do estado de execução, dá-se a escrita no *Register File*.

Cap14_P16_Estrutura, pág. 9

Nas instruções incluídas na *label* "isr", há uma série de instruções que se podem repetir. Estas ocorrem se não se obteve um valor igual ou inferior a 20(num valor de 5 bits). Logo, em 2^5 -1 = 31 hipóteses. A probabilidade de ser inferior a 20 é: 20/31 = 0.645. Podemos arredondar este valor para 2/3. Portanto, a hipótese de ser preciso arranjar um valor outra vez é de 1/3. Logo, vamos assumir que se chama "getvalue" apartir de "blt endisr" uma vez. Também contamos as instruções contidas em "getvalue", ao chamarmos esta *label*

Tempo: 28 instruções * 3 = 84 ciclos. 84*0.002 = 0.168s

Conclusão:

Com a realização deste trabalho, foi compreendida a enorme importância da cadeira de Arquitetura de Computadores neste curso. Compreender linguagem de baixo nível, ligações, manipulação de bits, etc. é extremamente importante para compreender melhor os computadores.

A ideia de que, com circuitos e operações, que à partida podem parecer muito simples, se consegue fazer muito mais do que se pensa, é algo que gostei muito de experienciar e aprender. Que aquilo que parece complexo, na verdade é composto por imensas construções muito simples, eficazes e lógicas.

Bits, as suas manipulações, as várias ligações e eletrónica, revelam-se como a base fundamental da imensa tecnologia que existe hoje em dia. E faz-nos pensar quão longe já viemos. Só com este pequeno jogo, foi preciso algum planeamento, circuitos e testes. E hoje em dia, já temos telemóveis, como computadores portáteis, resistentes e duráveis, que cabem no nosso bolso. Por isso, para além do que foi dito anteriormente, esta cadeira também nos dá perspetivas sobre a tecnologia que temos hoje em dia.

No que diz mais respeito ao trabalho, achámos que foi muito importante fazer testes simples e progressivos, que foi o que fizemos. Começámos só por entender muito bem o SASO timer e depois implementámos o resto a um bom ritmo, vindo a compreender todos os passos e possibilidades. Tentámos simplificar o código o máximo possível, escrevendo-o o mais logicamente possível.

Agradecimentos:

Professor Tiago Dias

Professor Mário Simões

Referências:

PDS16 Quick Reference, SDP16 User Manual V2.0

P16 – Manual de consulta rápida [das instruções] (v0.2)

SDP16-Schema, Revision 1.3

SASO Timer v3 datasheet

Folhas de apoio o prof. Paraíso

https://components101.com/7-segment-display-pinout-working-datasheet

Listagens dos programas realizados (.lst)

Fase 1(sem interrupção):

P16 assembler v1.2.2 (May 24 2019) fase1.lst Fri Jan 17 23:26:27 2020

Sections

Index	Name	Addresses	Size
0	.startup	0000 - 0007	0008 8
1	.text	0008 - 0027	0020 32
2	.data	0028 - 0027	0000 0
3	.stack	0028 - 0067	0040 64

Symbols

Name	Type	Value	Section
line#22	LABEL	0014 20	.text
saso_addr_temp	LABEL	0026 38	.text
getvalue	LABEL	001A 26	.text
start	LABEL	0002 2	.startup
line#10	LABEL	0006 6	.startup
loopgetvalue	LABEL	0016 22	.text
stack_top	LABEL	0068 104	.stack
main	LABEL	0008 8	.text
SASO_ADDR	ABSOLUTE	FF40 65344	.startup
timer_init	LABEL	0020 32	.text

Code listing

```
 1                      .equ SASO_ADDR, 0xFF40 ;Considerando que o periférico SASO_Timer_v3 deverá ser acessível na gama de endereços 0xFF40
a 0xFF7F
 2
 3
 4              .section .startup
 5 0000 0058                    b start
 6
 7              start:
 8 0002 8D66                    mov sp, stack_top
 9 0004 015C                    bl main
10 0006 FF5B                    b .
11
12                     .text
13             main:
14 0008 0E24           push lr
15
```

```
16 000A 8060        mov r0,8
17 000C 095C        bl timer_init
18 000E 055C        bl getvalue
19
20 0010 A10C        ldr r1, saso_addr_temp
21 0012 015C        bl loopgetvalue
22 0014 FF5B        b .
23
24          loopgetvalue:                       ;uint8_t timer_get_value(void);          3
25
26 0016 1008                ldrb r0, [r1]
27 0018 FE5B                b loopgetvalue
28
29          getvalue:                           ;uint8_t timer_get_value(void);          3
30 001A 510C                ldr r1, saso_addr_temp
31 001C 1008                ldrb r0, [r1]
32 001E 0FB7                mov pc, lr
33
34
35          timer_init:                         ;void timer_init(uint8_t interval)       5, iniciar o LR, para comecar a contar de 1
certo valor
36 0020 210C                ldr r1, saso_addr_temp
37 0022 1028                strb r0,[r1]
38 0024 0FB7                mov pc,lr
39
40
41
42          saso_addr_temp:
            ;tem q tar sempre a frente para ser usado
43 0026 40FF                .word       SASO_ADDR
44
45                  .data
46
47                  .section .stack
48 0028 00000000            .space 64
48 002C 00000000
48 0030 00000000
48 0034 00000000
49          stack_top:
50
```

Fase 1(com interrupção)

P16 assembler v1.2.2 (May 24 2019) fase1intrup.lst Fri Jan 17 23:26:35 2020

Sections

Index	Name	Addresses	Size
0	.startup	0000 - 0009	000A 10
1	.text	000A - 0037	002E 46
2	.data	0038 - 0037	0000 0
3	.stack	0038 - 0077	0040 64

Symbols

Name	Type	Value	Section
saso_addr_temp	LABEL	0036 54	.text
getvalue	LABEL	002A 42	.text
line#20	LABEL	0014 20	.text
enableisr	LABEL	0016 22	.text
start	LABEL	0004 4	.startup
timer_init	LABEL	0030 48	.text
SASO_ADDR	ABSOLUTE	FF40 65344	.startup
isr	LABEL	0022 34	.text
stack_top	LABEL	0078 120	.stack
main	LABEL	000A 10	.text
line#11	LABEL	0008 8	.startup

Code listing

```
  1                     .equ SASO_ADDR, 0xFF40 ;Considerando que o periférico SASO_Timer_v3 deverá ser acessível na gama de endereços 0xFF40
a 0xFF7F
  2
  3
  4                 .section .startup
  5 0000 0158                   b start
  6 0002 0F58                   b isr
  7
  8                 start:
  9 0004 8D67                   mov sp, stack_top
 10 0006 015C                   bl main
 11 0008 FF5B                   b .
 12
 13                    .text
 14                main:
 15 000A 8060           mov r0,8
 16 000C 115C           bl timer_init
 17 000E 035C           bl enableisr
 18 0010 0E04           pop lr
```

```
19 0012 0FB0        mov pc, r0

20 0014 FF5B        b .

21

22          enableisr:

23 0016 6CB0                mrs r12, cpsr

24 0018 50B6                msr spsr, r12                                   ;salvar flags

25 001A 0061                mov r0, 0b10000                 ;IE = 1

26 001C 40B0                msr cpsr,r0                                     ;automaticamente atualiza LR para a instrucao a seguir desta,
aqui as flags podem ser alteradas

27 001E 40B6                msr cpsr, r12                                   ;recoperamos as flags, IE = 0 automaticamente


28 0020 FA5B                b enableisr

29

30          isr:

31 0022 0E24                push lr

32 0024 025C                bl getvalue

33 0026 0E04                pop lr

34 0028 0FB7                mov pc, lr

35

36

37          getvalue:                                       ;uint8_t timer_get_value(void);         3

38 002A 510C                ldr r1, saso_addr_temp

39 002C 1008                ldrb r0, [r1]

40 002E 0FB7                mov pc, lr

41

42

43          timer_init:                                     ;void timer_init(uint8_t interval)      5, iniciar o LR, para comecar a contar de 1
certo valor

44 0030 210C                ldr r1, saso_addr_temp

45 0032 1028                strb r0,[r1]

46 0034 0FB7                mov pc,lr

47

48

49          saso_addr_temp:
            ;tem q tar sempre a frente para ser usado

50 0036 40FF                .word       SASO_ADDR

51

52                  .data

53

54                  .section .stack

55 0038 00000000            .space 64

55 003C 00000000

55 0040 00000000

55 0044 00000000

56          stack_top:
```

Fase 2(encurtei uns comentários para não haver muitos intervalos entre as linhas, melhorar o formato)

Sections

Index Name Addresses Size

0 .startup 0000 - 0009 000A 10

1 .text 000A - 010B 0102 258

2 .data 010C - 0121 0016 22

3 .stack 0122 - 1121 1000 4096

Symbols

Name Type Value Section

array_shape_numb_end LABEL 011C 284 .data

array_shape_numb LABEL 010C 268 .data

port_addr LABEL 010A 266 .text

saso_addr_temp LABEL 00FC 252 .text

button1 LABEL 002C 44 .text

storechoice LABEL 0062 98 .text

port_input LABEL 00FE 254 .text

getvalue LABEL 00F0 240 .text

button0 LABEL 001C 28 .text

stack_top LABEL 1122 4386 .stack

main LABEL 000A 10 .text

port_output LABEL 0104 260 .text

SDP16_PORTS_ADDRESS ABSOLUTE FF00 65280 .startup

spin LABEL 003A 58 .text

init LABEL 0010 16 .text

SASO_ADDR ABSOLUTE FF40 65344 .startup

waitime LABEL 011E 286 .data

whiledelay LABEL 0080 128 .text

timer_init LABEL 00F6 246 .text

stack_top_addr LABEL 0008 8 .startup

componto LABEL 005A 90 .text

cycle LABEL 003E 62 .text

delay LABEL 007C 124 .text

semponto LABEL 005C 92 .text

enableisr LABEL 008A 138 .text

TIME_DELAY ABSOLUTE 00FA 250 .startup

```
correct          LABEL   00D8 216   .text
endisr           LABEL   0096 150   .text
chosen_addr      LABEL   007A 122   .text
restart          LABEL   00E4 228   .text
start            LABEL   0004 4     .startup
random_addr      LABEL   00D6 214   .text
getagain         LABEL   00A6 166   .text
addr_array       LABEL   011C 284   .data
chosen           LABEL   0120 288   .data
whiledelay_end   LABEL   0086 134   .text
isr              LABEL   00A0 160   .text
waittime_addrs   LABEL   00C0 192   .text
randomnumber     LABEL   0121 289   .data
draw             LABEL   00C2 194   .text
chosen_addrr     LABEL   00EE 238   .text
incorrect        LABEL   00E0 224   .text
```

Code listing

```
 1                      .equ SASO_ADDR, 0xFF40 ;Considerando que o periférico SASO_Timer_v3 deverá ser acessível na gama de endereços 0xFF40 a 0xFF7F
 2                      .equ SDP16_PORTS_ADDRESS, 0xFF00
 3                      .equ TIME_DELAY, 250
 4
 5              .section .startup
 6 0000 0158            b start
 7 0002 4E58            b isr
 8
 9              start:
10 0004 1D0C                   ldr sp, stack_top_addr
11 0006 0158            b main
12
13              stack_top_addr:
14 0008 2211            .word stack_top
15
16                     .text
17              main:
18              ;push lr
19 000A F06F            mov r0, 0b11111111
20 000C 745C            bl timer_init       ; inicar o saso timer com 1 valor para comecar a contagem descrescente
```

```
21 000E 0058    b init
22
23              init:
24 0010 6F5C            bl getvalue
25 0012 0268        mov r2, 0b10000000    ;get value subtitui r0 e r1
26 0014 00C1        and r0, r0, r2      ;mask most significant bit
27 0016 765C        bl port_output              ;acao no ponto, (piscar)
28 0018 015C        bl button0          ;confirmar ativacao i0
29 001A FA5B            b init
30
31              button0:                        ;PRESSIONAMENTO BOTOES  I0
32 001C 0E24            push lr
33 001E 1260        mov r2, #1          ;r2->mask
34 0020 6E5C            bl port_input               ;r0->valor lido
35 0022 00C1        and r0, r0, r2      ;r0 & mask -> valor para avaliar, ver se e um 0 ou 1
36 0024 0060        mov r0,#0           ; spin(0) (sem ponto)
37 0026 0944        bzc spin            ;if flag Z = 0, o botao esta pressionado, e vai para o spin
38 0028 0E04        pop lr                          ;se nao, volta para o ciclo, do ponto piscar
39 002A 0FB7            mov pc,lr
40              button1:            ;I1
41 002C 0E24            push lr
42 002E 2260        mov r2, #2
43 0030 665C        bl port_input
44 0032 00C1        and r0, r0, r2      ;(r0->mask(um "adress")
45 0034 1644        bzc storechoice     ;if flag Z = 0, o botao esta pressionado, e vai para o storechoice
46 0036 0E04        pop lr                          ;se nao, volta para o ciclo, de estar a rodar
47 0038 0FB7            mov pc,lr
48
49              spin:   ;spin(bit mode)  mode = 0, ponto desligado, mode = 0x80(1000 0000), ponto ligado
50 003A 1560            mov r5,#1
51 003C 0464        mov r4,#64          ; vou de 00000001 para 00100000 com shifts -> 2^6 = 64
52              cycle:
53 003E 06B0            mov r6, r0          ;r6 tem o modo, o bit que define se e com ponto ou nao
54 0040 0160        mov r1,#0           ;reset, apagar tudo, mais simples.
55 0042 10CB            or r0, r1, r6               ;para suportar, ou com o ponto ligado ou desligado
56 0044 5F5C            bl port_output
57 0046 1A5C            bl delay
58 0048 50CB            or r0, r5, r6       ;para suportar, ou com o ponto ligado ou desligado
```

```
59 004A 5C5C            bl port_output          ;acender o q é para acender, acender o proximo
60 004C 175C            bl delay
61 004E D5E0            lsl r5,r5,#1    ;muito importante
62 0050 60A8            sub r0, r6, #0   ;ver se é spin com ponto ou nao
63 0052 00B3        mov r0,r6                   ;recuperar o modo
64 0054 0244        bzc componto
65 0056 EA5F            bl button1      ;ver se I1 esta carregado
66 0058 0158        b semponto
67          componto:
68 005A 1744            bzc enableisr
69          semponto:
70 005C 50BA            cmp r5,r4               ;for(int i(r5)=1; i<64; i<<1) i<64(decimal)
71 005E ED43            bzs spin        ;recomecar o ciclo
72 0060 EE5B            b cycle         ;o ciclo ainda nao acabou, continuar o ciclo
73
74          storechoice:
75 0062 4D5C            bl port_input
76 0064 016F        mov r1, 0b11110000  ;mask i7-i4
77 0066 80C0        and r0, r0, r1
78 0068 00EA            lsr r0, r0, #4
79 006A 710C            ldr r1, chosen_addr
80 006C 1028        strb r0, [r1]
81 006E 405C            bl getvalue                         ;gerar numero aleatorio
82 0070 00EA            lsr r0, r0, #4                  ;apropriar para ser 4 bits
83 0072 110F        ldr r1, random_addr
84 0074 1028        strb r0,[r1]
85 0076 0068        mov r0,#128                 ;continuar a fazer spin, mas com ponto agora
86 0078 F15B            b semponto                  ;continuar o ciclo de onde estava
87
88          chosen_addr:
89 007A 2001            .word chosen
90
91          delay:
92 007C 0E24            push lr
93 007E A16F            mov r1, TIME_DELAY
94              ;mov r1, TIME_DELAY & 0xff  ;se for uma valor mais elevado, usa-se isto
95              ;movt r1, TIME_DELAY >> 8
96          whiledelay:
```

```
97 0080 0240            bzs whiledelay_end
98 0082 91A8            sub r1, r1, #1                  ; && --delay_counter > 0
99 0084 FD47            bzc whiledelay
100             whiledelay_end:
101 0086 0E04           pop lr
102 0088 0FB7           mov pc, lr
103
104             enableisr:              ;(IRQ)
105 008A 6CB0           mrs r12, cpsr                   ;3ciclos
106 008C 0061           mov r0, 0b10000         ;3c
107 008E 40B0           msr cpsr,r0     ;automaticamente atualiza LR para a instrucao a seguir desta
108 0090 40B6           msr cpsr, r12                   ;recoperamos as flags
109 0092 0068           mov r0,#128             ;continuar a fazer spin, mas com ponto agora
110 0094 E35B           b semponto              ;faço branch para "semponto" mas é só pq assim é
111             endisr:                         ;end bet
112 0096 40B6           msr cpsr, r12
113 0098 4361           mov r3,#20
114 009A 220D           ldr r2, waittime_addrs
115 009C 2320           str r3,[r2]             ;reset variavel tempo, contagem de tempo, 5s, 5s->20ticks
116 009E 1158           b draw
117             isr:
118 00A0 0E24           push lr
119 00A2 E10C           ldr r1, waittime_addrs
120 00A4 1300           ldr r3,[r1]
121             getagain:
122 00A6 245C           bl getvalue             ;geracao numero aleatiorio,
123 00A8 5261           mov r2, 0b00010101              ;21
124 00AA F161           mov r1, 0b00011111
125 00AC 80C0           and r0,r0,r1    ;random val entre 0-20 ticks, que faz variar um valor X entre 20-40
126 00AE 00B9           cmp r0,r2                               ;random val->r0, entre 0 e 20
127 00B0 FA4F           bhs getagain                    ;confirmar q é menor que 21

128 00B2 3080           add r0,r3,r0                    ;random range
129 00B4 F057           blt endisr      ;se a soma anterior deu numero negativo, salta, salta se N = true
130             ;se o valor < 0, é pq fomos de 5s + (valor aleatorio neste instante) até valor inferior a 0, logo concluimos a contagem de pelo menos 5s mais um valor aleatorio(e q nao é superior a 5s), a menos q 0, logo terminamos a contagem aleatoria
131 00B6 B3A8           sub r3,r3,#1                    ;dec do fixo
132 00B8 310C           ldr r1, waittime_addrs
```

```
133 00BA 1320                   str r3,[r1]                             ;store do fixo decrmentado
134 00BC 0E04                   pop lr
135 00BE 0FB7                   mov pc, lr
136
137             waittime_addrs:
138 00C0 1E01                   .word waitime
139
140             draw:                                                   ;draw(numero q calhou) numero que foi
sorteado na roleta
141 00C2 900C                   ldr r0, random_addr
142 00C4 0508                   ldrb r5,[r0]                            ;r5->numero q calhou
143 00C6 A10E                   ldr r1, addr_array              ;base
144 00C8 9182                   add r1,r1,r5                            ;soma para escolher o o correspondente
145 00CA 1008                   ldrb r0, [r1]                           ;shape do numero que saiu
146 00CC 010D                   ldr r1, chosen_addrr
147 00CE 1308                   ldrb r3, [r1]                           ;numero escolhido
148 00D0 B0BA                   cmp r3,r5
149 00D2 0240                   beq correct
150 00D4 0558                   b incorrect
151             random_addr:
152 00D6 2101                   .word randomnumber
153             correct:                        ;menor no shape um ponto se ta certo, sem ponto se esta incorreto
154 00D8 0168                   mov r1,#128
155 00DA 80C8                   or r0,r0,r1
156 00DC 135C                   bl port_output
157 00DE 0258                   b restart
158             incorrect:
159 00E0 115C                   bl port_output
160 00E2 0058                   b restart
161             restart:
162 00E4 2260                   mov r2, #2      ;r2->mask
163 00E6 0B5C                   bl port_input           ;r0->valor lido
164 00E8 00C1                   and r0, r0, r2  ;r0 & mask -> valor para avaliar, ver se e um 0 ou 1
165 00EA 9247                   bzc init ;if flag Z = 0, o botao esta pressionado, e vai para o init, para recomecar o jogo
166 00EC FB5B                   b restart
167
168             chosen_addrr:
169 00EE 2001                   .word chosen
170
```

```
171                 getvalue:                                   ;uint8_t timer_get_value(void);     read value do timer
172 00F0 510C                   ldr r1, saso_addr_temp
173 00F2 1008                   ldrb r0, [r1]
174 00F4 0FB7                   mov pc, lr
175
176                 timer_init      ;void timer_init(uint8_t interval)  write value para o timer. 5, iniciar o LR, para
177 00F6 210C                   ldr r1, saso_addr_temp
178 00F8 1028                   strb r0,[r1]
179 00FA 0FB7                   mov pc,lr
180
181                 saso_addr_temp:

182 00FC 40FF                   .word     SASO_ADDR
183
184                 port_input:                                         ;ler do port
            ;PORT OPERATIONS
185 00FE 510C                   ldr r1, port_addr
186 0100 1008                   ldrb r0, [r1]
187 0102 0FB7                   mov pc, lr
188                 port_output:                                ;escrever
189 0104 210C                   ldr r1, port_addr
190 0106 1028                   strb r0, [r1]
191 0108 0FB7                   mov pc, lr
192
193                 port_addr:
194 010A 00FF               .word SDP16_PORTS_ADDRESS
195
196                 .data
197                 array_shape_numb:
198 010C 3F065B4F               .byte 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x77, 0x7C,
0x39, 0x5E, 0x79, 0x71
198 0110 666D7D07
198 0114 7F6F777C
198 0118 395E7971
199                             ;    0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
200                 array_shape_numb_end:
201
202                 addr_array:
203 011C 0C01                   .word array_shape_numb
```

```
204             waitime:
205 011E 1400           .word 20        ;inicializar a contagem do intervalo aleatorio apartir de 5s(20 ticks),
206             chosen:
207 0120 00         .byte 0
208             randomnumber:
209 0121 00         .byte 0
210
211             .section .stack
212 0122 00000000           .space 4096
212 0126 00000000
212 012A 00000000
212 012E 00000000
213             stack_top:
214
215
216
```