**Estruturação projeto IAC**

* **Inicializações gerais:**
  + Inicialização do SP e do BTE (com a tabela de interrupções).
  + Permissão das interrupções (enable interruptions);
  + Inicialização do contador de pontos a zero
  + Limpeza do screen e colocação da barra vertical
  + Ativação da flag para começar a gerar peças (tetraminós)
  + Jogo é inicializado apos carregar a tecla de start (espera\_start). Este ciclo vai servir também para reiniciar o jogo caso o utilizador assim o deseje
  + Inicializa-se um ciclo infinito que faz com que o jogo seja contínuo, até o utilizador o terminar (STOP).
* **Tecla\_pressionada:**
  + Vai servir para avaliar o valor da tecla pressionada e o local onde foi pressionada, linha e coluna correspondente na matriz do teclado. Isto vai permitir que saibamos que tecla foi carregada e qual foi carregada, através de um varrimento constante.
* **Limpa pixelscreen:**
  + Com o valor máximo vertical, ou seja, qual a ultima linha em hexadecimal (20H), vai se inicializar uma função que permita correr todos os pixéis do screen, pintando apenas os que pretendidos com a string 00, 08, 00, 00. Iremos realizar um ciclo que pinta todas as linhas, até atingir o final do pixelscreen. Isto vai-nos permitir limpar pixéis indesejados que já estavam no pixelscreen, implementado logo a barra vertical.
* **Print tetraminó:**
  + Recebe uma word que contem a peça tetramino atual, ou seja, em conjunto com o ciclo gerador de tetraminos, vamos fazer print do tetramino no ecrã. Os tetraminos são desenhados com as dimensões das linhas e colunas primeiro e depois os pixéis a pintar ou não pintar em cada linha em STRINGS. R10 que contém a posição inicial do tetramino 0004H. Vamos utilizar uma máscara que nos põe em evidência dos bits de menor peso e posteriormente os bits de maior peso. Para descobrir a linha e a coluna em que vamos escrever, já que todos os números possuem uma parte y,x. A existência de um ciclo que pinta ou não pinta, depende do desenho do tetraminó. A existência de um contador com o numero de colunas serve para saber quando se deve implementar uma unidade quando se acaba de desenhar a coluna. O mesmo acontece com o contador das linhas, até terminar o desenho.
* **Pintar ponto:**
  + É com esta função que são pintados os pixéis no ecrã. Caso a flag do print tetramino seja encontrada ativa ele realiza esta função. Ela consiste em pintar o ponto numa determinada linha e numa determinada coluna, como assembly funciona com endereços hexadecimais teve que se arranjar uma maneira de conseguir escrever no local pretendido. (Nota: Faz-se SHR de R0 com 8 para obter só o y e o SHL para multiplicar por 4 (fórmula dada no enunciado)). Com a aplicação de uma máscara que isole o ultimo byte.
  + Este ciclo contem a função de testar se se pode escrever ou não o pixel (ativando a flag\_verificação), já que caso o local já tenha sido pintado, para não se pintar por cima. Então são utilizadas flags de verificação. Se a flags\_verificação se encontrar ativa, vai testar se se pode escrever o ponto ou não. Caso a flag verificação\_encontrada esteja ativa, não irá pintar.
* **Escreve movimento:**
  + Responsável pela queda do teraminó. É utilizado um valor simbólico para ter a certeza que foi carregado naquele comando.
  + No movimento da queda desativa-se a flag\_ponto que serve para puder realizar o movimento (print tetraminó). Após se pintar uma vez, realiza-se o mesmo para a linha seguinte. No entanto é necessário verificar se se pode escrever no local pretendido. Assim, iremos realizar o mesmo procedimento que fizemos com as flags no print do pixel.
    - Caso tenha sido encontrado irá comparar o valor do movimento com a LINHAMAISUM, se forem iguais existe a ativação da flag para criar uma peça.
    - Caso não tenha sido encontrado, desativa-se a flag, para repor o seu valor. E posteriormente desativa-se a flag\_verificação\_encontrada para poder escrever a nova posição sem problemas.
    - No final compara-se para ver o que se se continua a realizar o movimento.
* **Rodar peça:**
  + Quando a função é chamada, sabemos que teremos que apagar a posição atual do tetraminó e escrevê-lo numa nova posição/rotação.
  + É necessário registar qual a nova posição do tetraminó. Assim, iremos percorrer a tabela de words referentes às posições rotativas de cada tetraminó. (Quando o tetraminó chega à última posição possível, volta à primeira).
  + Na nova\_rotacao vai se incrementar a rotação querida caso não se tenha atingido o final da tabela de rotações.
  + Vai existir uma verificação se pode efetuar ou não a rotação.
    - Inicia-se a ativar a flag de verificação e chama-se a rotina que pinta o tetraminó.
    - Iremos desativar as flags de verificação e verificação\_encontrada. No entanto iremos guardar num registo o valor da flag\_verificação\_encontrada, isto irá servir para ver se a flag está ativa ou não, o que irá provocar, ou não, a nova rotação da peça.
      * Caso a flag esteja ativa, vai ativar a flag responsável por pintar um ponto e chamar a função responsável para pintar o tetraminó.
      * Caso contrário, vai subtrair dois ao índice da tabela atual, para voltar à rotação anterior. Posteriormente vai atualizar a memória.
* **Conversão pontuação:**

A STRING pontuacao\_disp\_hexa guarda o valor da pontuação em decimal. A função conversão\_pontuacao, transforma esse valor decimal em hexadecimal. Através da fórmula: (X//10)\*6+X, sendo X o valor decimal da pontuação. Após irá comparar se a pontuação não é igual à pontuação máxima, se for o valor 99 irá ser escrito no display. Se não for, o valor da pontuação atual, após a conversão irá ser escrito no display.

* **Teclado funções:**

Compara ciclicamente se alguma das teclas do jogo (0,1,2,5,6,E,F) foi pressionada.

* + Se a tecla 0 for pressionada, salta para a função move\_esq que confirma se o tetramino pode ir para a esquerda (se não bate na parede) e realiza esse movimento chamando a rotina escreve movimento.
  + Se a tecla 1 for pressionada, salta para a função move\_roda que irá chamar uma rotina que roda a peça.
  + Se a tecla 2 for pressionada, salta para a função move\_dto que realiza o movimento à direita através da rotina escreve movimento.
  + Se a tecla 5 for pressionada, salta para a função move\_baixo que utiliza a rotina escreve movimento para fazer a queda rápida do tetraminó.
  + Se a tecla 6 for pressionada, salta para a função pausa que deveria parar o jogo alterando o event\_1 (responsável pela descida do tetraminó), e ficando num ciclo infinito enquanto espera que outra tecla seja pressionada.
  + Se a tecla E for pressionada, o jogo é reiniciado, saltando para o start.
  + Se a tecla F for pressionada, o jogo entra em STOP e chama o limpa\_screen mas guarda o valor do display. Fica em loop à espera que outra tecla seja pressionada.
* **Gerador tetraminós:**
  + Gera um tetraminó aleatoriamente. Se a flag\_cria\_peca estiver ativa (a 1) é gerado um novo tetraminó. Adiciona 1 ao contador\_tetra, para ir gerando uma peça diferente, aleatoriamente. Se a flag estiver a 1 chama a função criar\_peca que irá gerar um novo tetraminó.
* **Criar peça:**
  + Função que põe em evidência os últimos dois bites da combinação da etiqueta, no entanto só queremos 4 valores (0-3) – peças, para ir incrementando para alterar qual a peça a escrever.
  + Nesta função é avaliada a criação do monstro com uma probabilidade de 25% (devido ao SHL R1,1; e ao CMP com 1 🡪 00 || 01 || 10 || 11).
  + Ativação da flag para criação do monstro vai permitir criar o monstro. Caso não seja ativado criar o monstro, o que vai fazer é realizar a rotina não\_monstro e irá criar uma peça tetraminó também – vai atualizar a memória.
  + (CMP R3,0 🡪 par o caso da montanha chegar lá em cima, e para o caso de eu querer pintar uma nova peça e não conseguir pintar poque já está ocupado, significa que se perdeu o jogo.)
* **Desce com clock:**
  + Primeira função interrupção que trata do descer automático da peça.
  + É ativado o event\_1 sempre que é gerada uma nova peça.
* **Interrupções**:
  + Rotina que trata de verificar se o evento está ativo. A sua verificação tem a consequência de escrever o movimento de queda. Assim a função, ‘escreve movimento’ é realizada após a interrupção.
  + O movimento lateral do monstro é realizado nesta rotina, da mesma forma que é feito a queda do tetraminó. No entanto, o seu movimento é feito noutra função.
  + No final tem que se repor o valor dos eventos.
* **Avalia** **linha**:
  + Função que verifica se a linha está completamente preenchida por tetraminós ou não.
  + Vai avaliar o que está escrito no pixelscreen com o valor de uma linha preenchida (0FFF8). Caso, sejam iguais vai limpá-la, caso contrário não limpa.
  + Quando uma linha não se encontra completa vai se avaliar a próxima, daí haver um contador com o numero de linhas existentes, quando chegar ao final, volta para o inicio. Quando se encontra uma linha completa, chama-se o ciclo limpar\_linha
* **Limpar\_linha:**
  + Faz descer as linhas, subtraindo o valor da linha, para esta descer e escrevê-la no local da anterior. Vamos realizar isto até ao topo no pixelscreen, no entanto na primeira linha vamos pintá-la com a string que pintámos a barra superior, mas apenas uma vez, já que só eliminamos uma linha de cada vez.
  + Neste ciclo também é chamado o ciclo que faz a pontuação, já que quando se apaga uma linha somam-se 5 pontos à pontuacao\_hexa\_display que terá de ser convertida novamente para hexadecimal.
* **Monstro:**
  + Vai-se escrever o desenho do monstro numa dada posição (1515H), tal como fizemos com os tetraminós.
* **Movimento esquerda do monstro:**
  + Interrupção que trata do movimento lateral do monstro, tal como fizemos com a queda das peças dos tetraminós, esta utiliza o clock 2.
* **Move monstro:**
  + Tal como no escreve movimento, o move\_monstro serve para realizar o movimento lateral do monstro.
  + Novamente é utilizada a mesma técnica de escrita de pixéis no ecrã.
  + (Ao utilizar o nibble 🡪 MASCARA\_LB com AND com a posição do monstro para ver se está a escrever ou não em cima da parede.). Caso não ache na parede continua a realizar movimento.