Escola Politécnica da Universidade de São Paulo

PCS3446 — Sistemas Operacionais

Primeira entrega

Monitor de Overlays

Alunos de graduação:
Pedro Henrique Lage Furtado de Mendonça 8039011

Docentes:
Prof. Dr. João José Neto

5 de Setembro de 2020

Conteúdo

Plataforma de desenvolvimento

Esse exercício foi desenvolvido no software **IDE68K** que apresenta um simulador para processadores da família Motorola 680XX tanto em forma de terminal quanto visual. O software também incluí uma ferramenta de desenvolvimento integrado com montador, compilador de C e outras regalias que facilitam o desenvolvimento tanto em assembly quanto em C.

O programa EX01_FillMemory.asm simplesmente preenche posições de memória com FF. Na figura 1 pode-se ver o simulador visual; ele permite visualizar e editar os registradores, a memória e a pilha, fazer execução step-by-step e inserir breakpoints. Também permite conectar periféricos como disco rígido, LEDs, switches, terminal, etc.

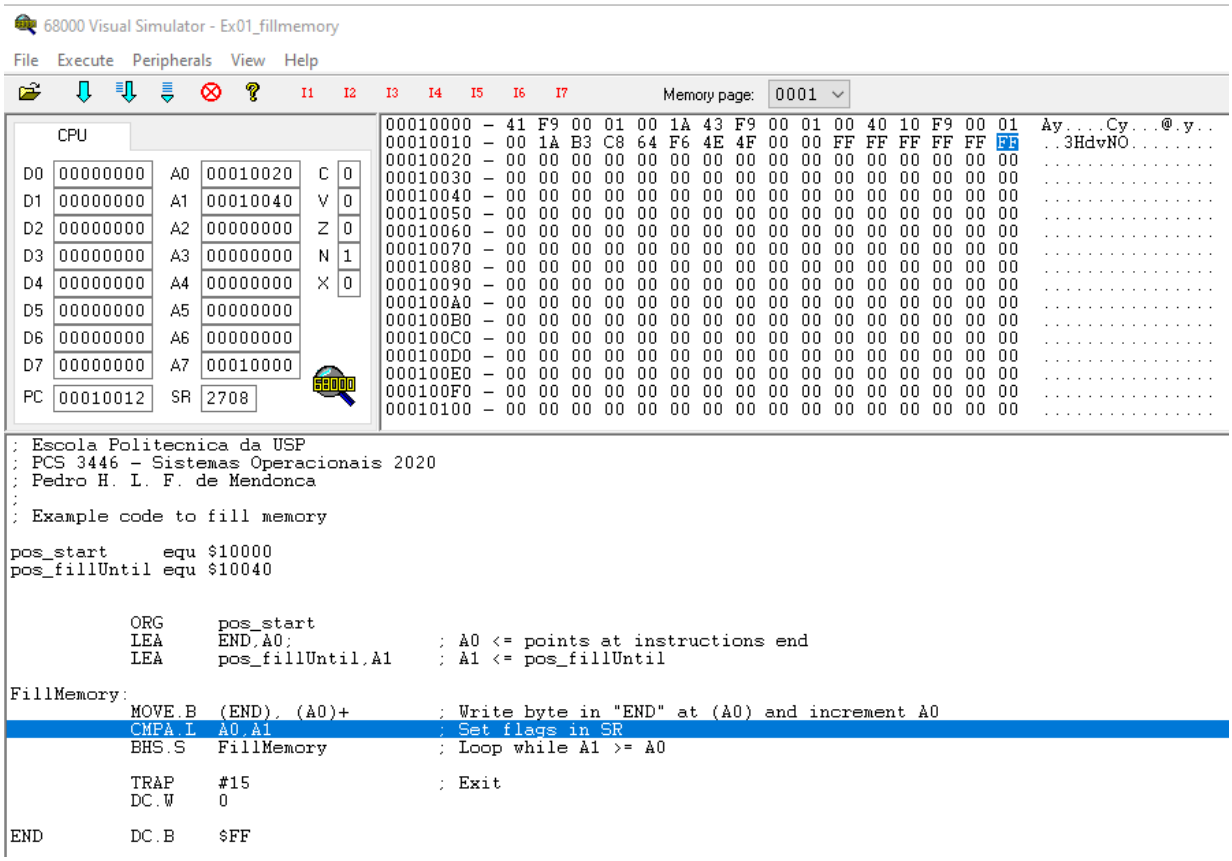

Figura 1: Simulador Visual durante execução do programa EX01_FillMemory

Já na figura 2 há uma captura do simulador em terminal. Ele tem algumas facilidades como trace (execução instrução a instrução), visualização de memória e registradores, edição de memória e registradores, etc. Na captura pode-se ver a memória antes de execução pelo comando DM (display memory) e após alguns ciclos a escrita de FF na posição 0x1001B.

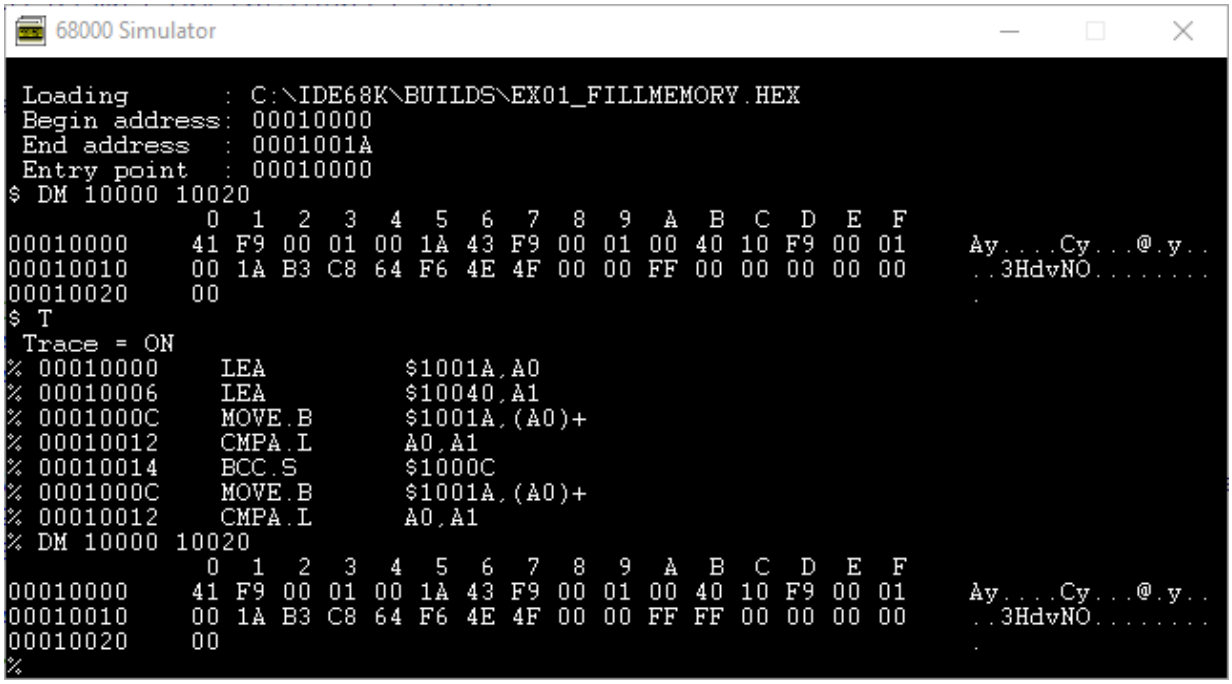

Figura 2: Simulador Terminal durante execução do programa EX01_FillMemory

Eu fiz a disciplina de sistemas de programação em 2017 e acredito que meu projeto de simulador de eventos discretos e a implementação da MVN seriam suficientes para a disciplina de Sistemas Operacionais mas optei pela plataforma IDE68K. O processador Motorola 68000 possui muito mais recursos que aquele implementado pela MVN porém não é um processador moderno (data de 1979) então ainda não contempla muitas das regalias de processadores mais novos. Além disso, todo código escrito por mim seria um código funcional em **um hardware de verdade** e poderia ser executado de fato no laboratório de processadores, por exemplo.

O fato de o processador M68000 ser mais poderoso que a MVN não invalida o esforço empenhado uma vez que eu não posso mexer em como o hardware funciona e devo desenvolver o trabalho em torno das limitações de um hardware de verdade. Isso não seria verdade com a MVN, se algo não desse certo, eu poderia modificar a implementação do nosso hardware fictício para facilitar ou até implementar funcionalidades em alto nível. Como o foco da disciplina é em sistemas operacionais, vejo mais benefício e sentido em desenvolver um código que seria funcional e executável fora de um simulador.

1 Kernel

Como o desenvolvimento desse trabalho se dá em cima de um hardware de verdade, tive de escrever um mini-kernel para construir o resto do sistema em cima dele. O processador M68000, além de interrupções, tem uma instrução chamada **TRAP**. Ao chamá-la, o processador entra em modo supervisor e salta para uma rotina de tratamento, idêntico ao que seria feito para interrupções. Cabe aqui distinguir TRAP como *exception*, gerada por software válido (ou seja, não houve erro de execução), enquanto que *interrupção* seria gerada por software inválido (violação de acesso de memória) ou por hardware.

O M68000 tem 16 possíveis chamadas de TRAP e a plataforma IDE68K toma a posição 16 (TRAP #15) para si. A plataforma então implementa algumas utilidades no TRAP #15, como PUTCH, GETCH, PRTSTR, etc. Dentre essas, ele também implementa um LOADER que carrega o arquivo apontado por A0 e o carrega com um offset dado por D0. Se o programa não for escrito de forma relocável, D0 deve ser zero. Sempre que possível, escrevi meus códigos em assembler zelando para que todo endereçamento fosse feito relativo à posição da instrução, assim, o código seria relocável.

1.1 TRAP #0 — Loader modificado

Apesar do IDE68K possuir um LOADER, me apropriei do TRAP #0 para implementar um LOADER modificado, a principal diferença é que o meu LOADER acrescenta tratamento de erro, ou seja, se não foi possível carregar o módulo do usuário, ele emite uma mensagem de erro e coloca o processador em estado de parada.

1.2 Alocação de espaço e proteção do Kernel

O kernel aloca espaço para o sistema operacional (de 0x400 a 0x600, equivalente a 512 bytes) e atualmente ocupa 144 bytes. Ele aloca 0x1000 bytes (4096 em decimal) para o usuário mas nenhum programa implementado chega perto disso, exagerei no tamanho para não correr nenhum risco de termos um stack overflow.

Além da alocação de espaço, o kernel também carrega o monitor de overlay em memória protegida do sistema operacional ocupando 0xB8 (184 em decimal) bytes adicionais e carrega o código *root* do usuário em memória do usuário e entrega o controle para ele fora do modo de supervisão. Assim, o usuário **não** pode manipular a memória que contém o SO e não pode interferir na máscara de interrupções.

1.3 TRAP #3 — Overlay monitor

Antes de carregar código do usuário, o kernel carrega o overlay monitor em posição de memória privilegiada e atribui ao monitor o TRAP #03, assim o usuário pode invocá-lo.

1.4 Código completo do Kernel

```
; Escola Politecnica da USP
; PCS 3446 - Sistemas Operacionais 2020
; Pedro H. L. F. de Mendonca
;
; Kernel Implementation
;
; Kernel system calls:
;    Trap #0 - Loader (A0 should point to module name as null terminated string and D0 will be overwritten with entry point)
;    Trap #3 - overlay monitor (A0 should point to module name as null terminated string, D0 should be module level and D1 module size)
;
; This kernel loads the overlay monitor (in TRAP #3) and the user root programs
; This implementation uses 44 bytes + overlay monitor
; User program must be at $10000 and is allocated $1000_hex (4096_dec) bytes.
;
;

; From 0x000 up to 0x3FF we have interruption tables and simulator data
; So our program starts at 0x400
            org     $400

; TRAP address'es are 0x80 + 4*vector value
trap0vec    equ     $80 ; Loader
; TRAP #3 = 0x8C
trap3vec    equ     $8C

; Set system stack pointer and loader in trap table
setup:
            LEA     sysSP, SP
            LEA     loader, A0
            MOVE.L  A0,trap0vec         ; Put address of loader into the trap vector table #0
            BRA.S   start

loader:
            ; A0 should contain address of module to load (as null terminated string)
```

```
35              ; D0 will be overwritten with loaded modules position in memory
36              TRAP    #15
37              DC.W    t15LOAD                 ; load child, entry point in D0
38              BEQ.S   loader_error            ; D0 = 0 on error
39              RTE
40      loader_error:
41              ; A0 already points to modules name
42              trap    #15
43              dc.w    t15PRTSTR               ; print modulename first
44              lea     errmsg,A0
45              trap    #15
46              dc.w    t15PRTSTR               ; followed by error message
47              trap    #15
48              dc.w    t15EXIT                 ; exit
49              stop    #$2700
50  errmsg  dc.b        ": error loading module",CR,LF,0
51
52  start:
53  ; Load overlay monitor
54              LEA     str_ovrlaymon,A0        ; get name of module to load
55              TRAP    #0                      ; Call loader
56              MOVE.L  D0,trap3vec             ; Put address of overlay monitor into
                    ↪ the trap vector table #3
57              SUB.L   D0,D0                   ; Make D0 = 0 so other loads don't
                    ↪ offset
58
59  ; Load user code
60              LEA     str_userModule,A0       ; get name of module to load
61              TRAP    #0                      ; D0 is loaded with beggining of user
                    ↪ program
62
63  ; Allocate space for user and jump to their code WITHOUT supervisor mode
64              MOVE.L  D0,-(SP)                ; Push D0 (beggining of user space)
                    ↪ in stack
65              MOVE.W  #$0,-(SP)               ; Push user SR in stack
66              ADDI.L  #usrMemSize,D0          ; Allocate usrMemSize for user (D0 =
                    ↪ usr beggining + usr mem size)
67              MOVE.L  D0,A0                   ; Allocate usrMemSize for user (A0 <=
                    ↪ D0)
68              MOVE.L  A0,USP                  ; Allocate usrMemSize for user (User
                    ↪ stack pointer <= A0) (USP <= D0 is invalid)
69              SUB.L   D0,D0                   ; Leave registers in known state for
                    ↪ user
70              SUB.L   A0,A0                   ; Leave registers in known state for
                    ↪ user
71              RTE                             ; Return from Exception (SR <= pop
                    ↪ stack = usr SR, PC <= pop stack = usr program)
72              ; RTE puts SR equal to last stack.W and PC equal to last stack.L
```

```
73              ; That's why we pushed 10000 and (SR without supervisor) into
                    ↪ stack.
74
75
76
77
78  sysSP       equ     $600
79  usrMemSize  equ     $1000               ; User SP points to here initially
80
81
82  ; Strings with null ending
83  str_ovrlaymon       dc.b        "overlay_monitor",0
84
85  str_userModule      dc.b        "OM01_root",0
86  ; ASCII characters
87  LF          equ     $0A
88  CR          equ     $0D
89  ; TRAP #15 Codes
90  t15PRTSTR       equ     7
91  t15EXIT         equ     0
92  t15PRTNUM       equ     5
93  t15GETNUM       equ     6
94  t15LOAD         equ     19
```

2 Overlay Monitor

2.1 Como usar?

O usuário deve apontar A0 para uma string terminada em NULL. Essa string deve conter o nome do módulo a ser carregado. Em *D0.BYTE* deve constar o nível do módulo a ser carregado. Por fim, em *D1.BYTE* deve constar o tamanho do módulo a ser carregado.

O usuário pode carregar até *maxUserLvl* módulos simultâneamente, definido por padrão como 5; Como especificado acima, o nível de carregamento (visualizando os overlays em estrutura de árvore, o nível seria distância do root. Além do root o usuário pode carregar 5 outros módulos) é dado pelo *D0.BYTE* e se D0 for 0 ou maior que maxUserLvl o monitor dará mensagem de erro e irá parar o processador.

Vale ressaltar que o monitor de overlays **não** faz relocamento do código do usuário, ele o carrega onde o arquivo do usuário especifica. Ele assume como hipótese que, para um dado nível, **todos** os módulos daquele nível devem começar na mesma posição de memória. Ou seja, se o nível 2 começa na posição 0x10200, **todos** os módulos de nível 2 devem começar na posição 0x10200. Isso foi feito para poder garantir que se um módulo de 300 bytes fosse carregado e, em seguida um de 200, essa diferença de 100 bytes seria limpa pelo monitor de overlay, sem deixar a memória "suja" de código desconhecido.

2.2 Funcionamento

Primeiramente, todo código do monitor foi escrito de forma relocável usando o PC como modo de endereçamento relativo. Assim, o overlay monitor pode ser carregado pelo loader utilizando um offset.

Um ponto importante que devemos nos atentar é que o overlay monitor não funciona sozinho. Ele é dependente do kernel escrito anteriormente pois ele utiliza o loader implementado em TRAP #00. Também é interessante notar que, apesar de ele ser carregado em memória privilegiada do sistema operacional, ele não utiliza nenhuma instrução ou posição de memória que obrigue ele a estar dentro do SO, assim, ele pode ser carregado pelo usuário em vez de ser carregado pelo kernel e, em vez de chamá-lo pelo TRAP #03, pode-se chamá-lo como uma subrotina normalmente. (Só deve-se trocar a instrução RTE (*return from exception*) na linha 77 por RTS (*return from subroutine*)

Passos de execução:

1. **Registradores do Usuário** — O overlay monitor salva os registradores do usuário antes de executar, assim, se não houver erro na execução, o usuário recebe seus registradores da mesma forma que o entregou.

2. **Inicialização** — O overlay monitor tem uma tabela de tamanho *maxUserLvl* bytes, definido por padrão como 5. Na primeira chamada do monitor ele roda uma rotina de inicialização de forma a escrever 0x00 em todas as posições dessa tabela. Essa tabela serve para guardar o tamanho do módulo atualmente carregado na memória.

3. **Teste de compatibilidade** — Verifica de D0.byte se encontra entre 1 e maxUserLvl. Se estiver fora, uma mensagem de erro é exibida e o processador é parado.

4. **Diferença de tamanho dos módulos** — Dado o nível em D0, busca-se a posição da tabela que aponta para o tamanho do módulo carregado **atualmente**. Se esse tamanho for **maior** que o módulo que será carregado em seguida, a diferença de tamanhos é armazenada em D7.

5. **Carregamento do módulo novo** — O módulo novo é carregado na memória

6. **Remoção de dados remanescentes** — Se o módulo novo for menor que o módulo antigo a diferença de tamanhos está guardada em D7. Então nessa etapa o processador vai para a posição final do módulo antigo e apaga byte por byte, usando D7 como controle de laço, até chegar no fim do módulo novo. Assim, toda memória que continha pedaços remanscentes do módulo antigo é sobreescrita com 0x00.

7. **Registradores do Usuário** — Recupera-se os registradores do usuário

8. **Return** — retorna o processador ao código do usuário que chamou o TRAP #03

2.3 Testes

Durante o desenvolvimento testou-se cada "macro-bloco" de lógica exaustivamente, testou-se variar o tamanho da tabela, ter certeza que a inicialização estava esvaziando a tabela de forma apropriada, que a montagem estava sendo feita com endereços relativos ao PC para que o código fosse relocável, testou-se todas as condições de branch para ter certeza que os desvios não fossem tomados em momentos inadequados e testou-se a funcionalidade de apagar memória remanescente de um módulo maior que o último a ser carregado e também testou-se o carregamento de um a cinco módulos simultâneamente.

2.4 Código completo do Overlay Monitor

```
1 ; Escola Politecnica da USP
2 ; PCS 3446 - Sistemas Operacionais 2020
3 ; Pedro H. L. F. de Mendonca
4 ;
5 ; Overlay monitor implementation
```

```
6  ; Overlay monitor has been implemented in such a manner that it is relocable
       ↪ ;
7  ;
8  ;
9  ; Overlay monitor will be called by user using TRAP
10 ; User must put address to module name (as null terminated string) in A0
11 ; D0.BYTE <= Module Level (1 to max)
12 ; D1.BYTE <= Module Size
13 ; User must obey maxUserLvl
14 ; Even using A0, D0 and D1 as parameters, they are NOT edited by the rotine
15 maxUserLvl equ      $5
16 maxUserLvlChar equ '5'
17
18
19           ORG      $4A0 ; Put overlay monitor at ... (it is relocable)
20
21 ; Save user registers
22           MOVEM.L D0/D1/D7/A1,-(SP)
23 ; Check if overlay monitor has been initialized
24           MOVE.B   (OM_init,PC), D7
25           BEQ.S    OM_initialize        ; Goto clear table if not initialized
26 OM_initialized:
27
28
29 ; Check if module level is compatible (Between 1 and maxUserLvl)
30           TST      D0                   ; Check if D0 == 0
31           BEQ.S    error_exceedLvl
32           CMPI.B   #maxUserLvl, D0
33           BHI.S    error_exceedLvl      ; Check if D0 > maxUserLvl (BHI is
                 ↪ unsigned, BGT is NOT)
34
35 ; Determine entry position in module size table
36           LEA      (Table,PC), A1       ; A1 points to table (index 0)
37           ADD.W    D0,A1                ; A1 points to user module position
                 ↪ +1
38           SUBQ.L   #1,A1                ; A1 points to correct user level
                 ↪ table position
39
40            ; A1 holds position to table(module_level)
41 ; If new module size < old module size, calculate memory difference
42           MOVE.B   (ZERO,PC),D7         ; Set memory difference to zero
43           CMP.B    (A1),D1
44           BHS.S    OM_LoadMod           ; size at Table+D0 <= D1, no need to
                 ↪ clear memory (BGE is unsigned, BHS is NOT)
45           MOVE.B   (A1),D7
46           SUB.B    D1,D7                ; D7 holds size difference of new
                 ↪ module to old module
47            ; A1 holds position to table(module_level)
```

```
48              ; D7 holds difference between module sizes (how many bytes to
                    ↪ clear)
49
50  OM_LoadMod:
51  ; Load new module
52              ; Save new module size in table
53              MOVE.B  D1,(A1)
54              SUB.L   D0,D0           ; Make D0 = zero so kernel loader doesn't
                    ↪ offset
55              ; A0 should contain address to module name, provided by user
56              TRAP    #0              ; Kernel loader
57              ; D0 will be set to program begginning
58
59  ; Clear memory
60              ; D7 holds difference between module sizes (how many bytes to
                    ↪ clear)
61              ; D0 is where the loaded module starts
62              TST D7
63              BEQ OM_restore          ; if D7 = 0, no need to clear memory
64          ; Here we have to use long bc D0 will probably be >= 10000 for user
                ↪ code
65              MOVE.L  D0,A1           ; A1 <= D0 = module start
66              ADD.L   D1,A1           ; A1 <= D0+D1 = module Start + module size
67              ADD.L   D7,A1           ; A1 <= D0+D1+D7 = module start + module size
                    ↪ + difference to old module
68              SUBI.L  #1,D7           ; Correction to index
69          OM_clearDiff:
70              MOVE.B  (ZERO,PC),-(A1)
71              SUBQ.B  #$01,D7
72              BNE.S   OM_clearDiff ; Clear while D7 != 0
73
74  ; Restore users registers and return
75  OM_restore:
76              MOVEM.L (SP)+,D0/D1/D7/A1
77              RTE
78
79  ; OM_initialize used A1 and D7
80  OM_initialize:
81              ; Loop through table writing zero in it
82              LEA     (OM_init,PC),A1
83              MOVE.B  (ONE,PC), (A1)      ; Set OM_init to 01
84              LEA     (Table,PC),A1
85              MOVE.B  (ZERO,PC), D7       ; D7 = zero
86              ADDI.B  #maxUserLvl, D7     ; D7 = maxUserlvl
87          OM_initialize_loop:
88              MOVE.B  (ZERO,PC), (A1)+    ; Erase content at (A1) and increment
                    ↪ A1
89              SUBQ.B  #$1, D7
90              BNE.S   OM_initialize_loop  ; Loop while D7 != 0
```

```
91              BRA.S    OM_initialized       ; return to overlay monitor call
92
93
94  error_exceedLvl:
95              lea      str_errUsrLvl,A0
96              trap     #15
97              dc.w     t15PRTSTR            ; print error msg
98              trap     #15
99              dc.w     t15EXIT              ; exit
100             stop     #$2700
101
102 OM_init     ds.b     1
103 ZERO        dc.b     $00
104 ONE         dc.b     $01
105
106 ; This table stores each levels max size
107 Table       ds.b     maxUserLvl
108
109 ; Strings with null ending
110 str_errUsrLvl        dc.b        "ERROR: User exceeded ",maxUserLvlChar, "
    ↪ levels or used level zero",CR,LF,0
111
112 ; ASCII characters
113 LF          equ      $0A
114 CR          equ      $0D
115
116 ; TRAP #15 Codes
117 t15PRTSTR      equ      7
118 t15EXIT        equ      0
119 t15PRTNUM      equ      5
120 t15GETNUM      equ      6
121 t15LOAD        equ      19
```

3 Código do Usuário

Para testar toda a composição de kernel + monitor de overlays, desenvolveu-se um código simples para simular o código de usuário.

3.1 User Root

O código root do usuário é carregado pelo kernel e posto em execução em modo usuário. O Root carrega o módulo A de nível 1 e o executa. Então carrega o módulo B de nível 1 e o executa. O módulo 1B é menor que o 1A, então o monitor de overlays deve limpar a diferença de memória entre eles.

Depois, o root carrega e executa o módulo 2.

Por fim, ele chama EXIT do processador que deixa-o parado.

3.2 Nível 1

3.2.1 Módulo A

Esse código preenche a memória de FF até o início do nível 2, ocupando 0x60 (96 em decimal) bytes. Ele também soma 0b00000001 ao registrador D6.

3.2.2 Módulo B

Esse código soma 0b00000010 ao registrador D6.

3.3 Nível 2

Esse código soma 0b00000100 ao registrador D6, carrega o módulo de nível 3 e o executa

3.4 Nível 3

Esse código soma 0b00001000 ao registrador D6.

3.5 Resultado

Ao fim, o código do usuário não produz nenhum resultado significante, mas, se todos os módulos forem chamados, D6 deve conter 0b00001111, ou seja, 0x0F. Também note que,

apesar dos módulos terem sido chamados em ordem (nível 1, 1, 2 e 3) isso não é necessário e pode-se carregar módulos de qualquer nível a qualquer momento, a única ressalva é, para um dado nível, o código sempre começa na mesma posição, ou seja, não é possível carregar dois módulos de um mesmo nível ao mesmo tempo.

3.6 Execução Completa

Para reproduzir essas etapas de execução, baixe o IDE68K, baixe todos os programas .asm presentes, abra eles um a um e aperte F7 para gerar a imagem hexadecimal de cada arquivo. É essa imagem hexa que o LOADER implementado carrega do disco. Então abra o kernel.asm e aperte F9 para executar no simulador visual ou CTRL+F9 para executar no simulador em terminal.

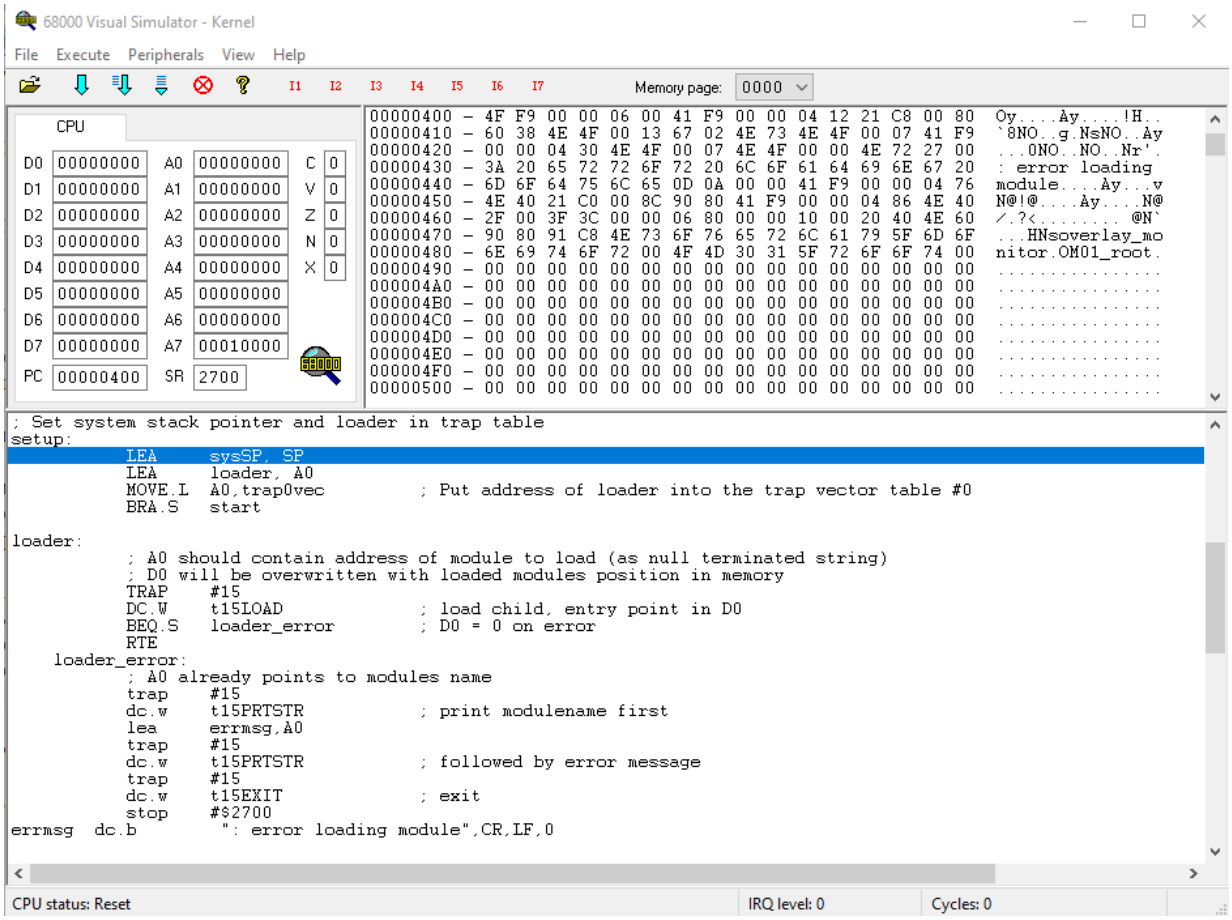

Figura 3: Apenas o kernel está na memória

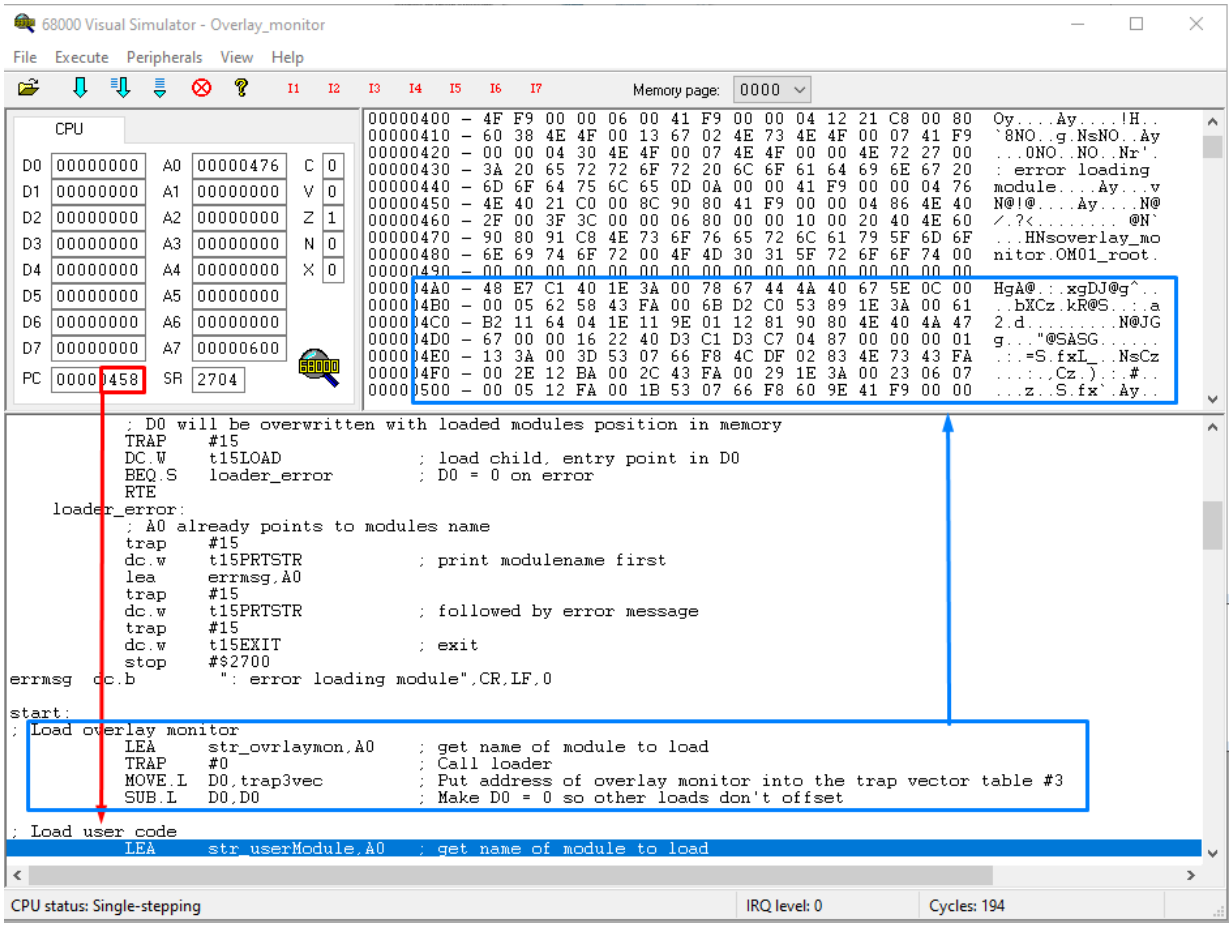

Figura 4: O overlay monitor foi carregado em 0x4A0

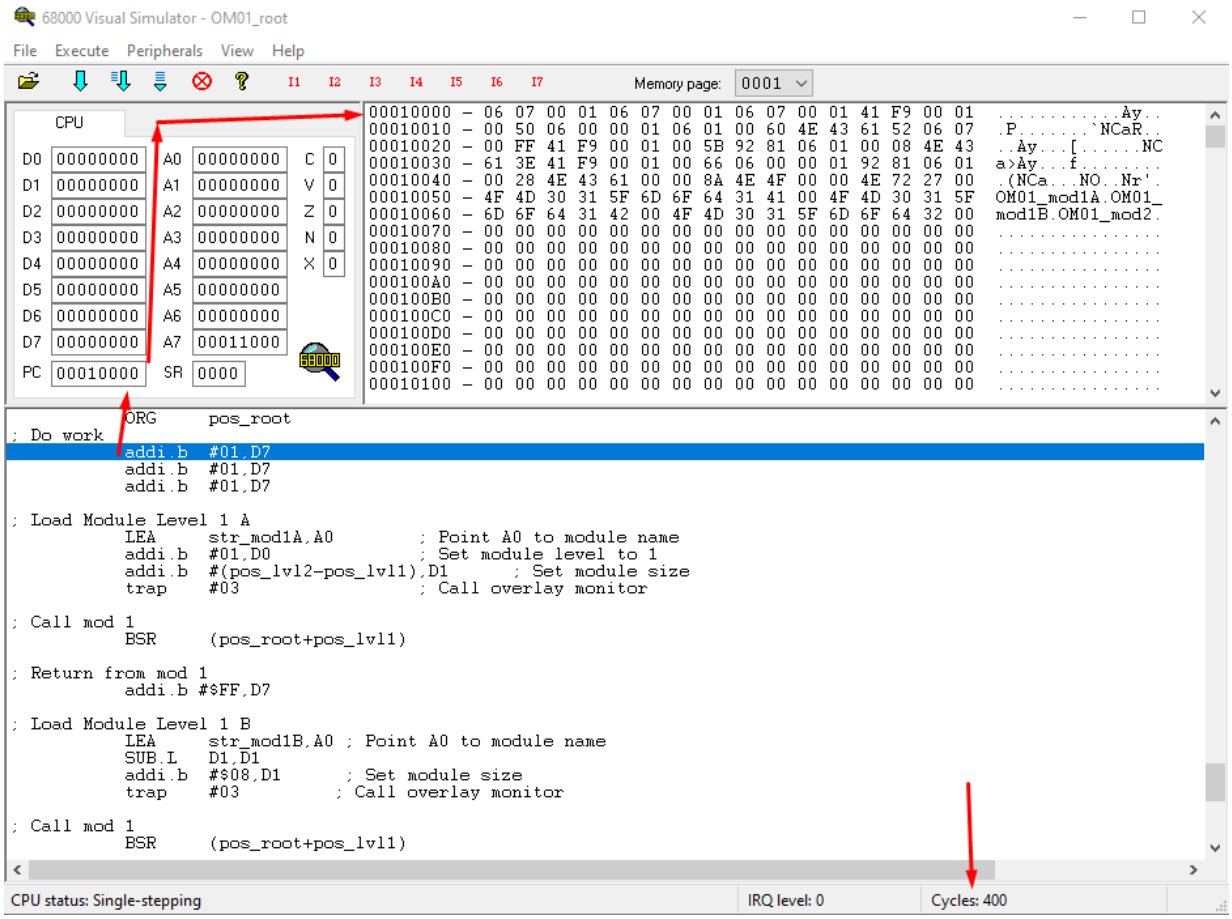

Figura 5: Código do usuário foi carregado e posto para executar em 0x10000.
Note que o SR marca 0x0000, ou seja, sem modo de supervisor (bit 13) e sem máscara de interrupção (bits 8 a 10)
Note também que as posições além de 0x10070 estão vazias.

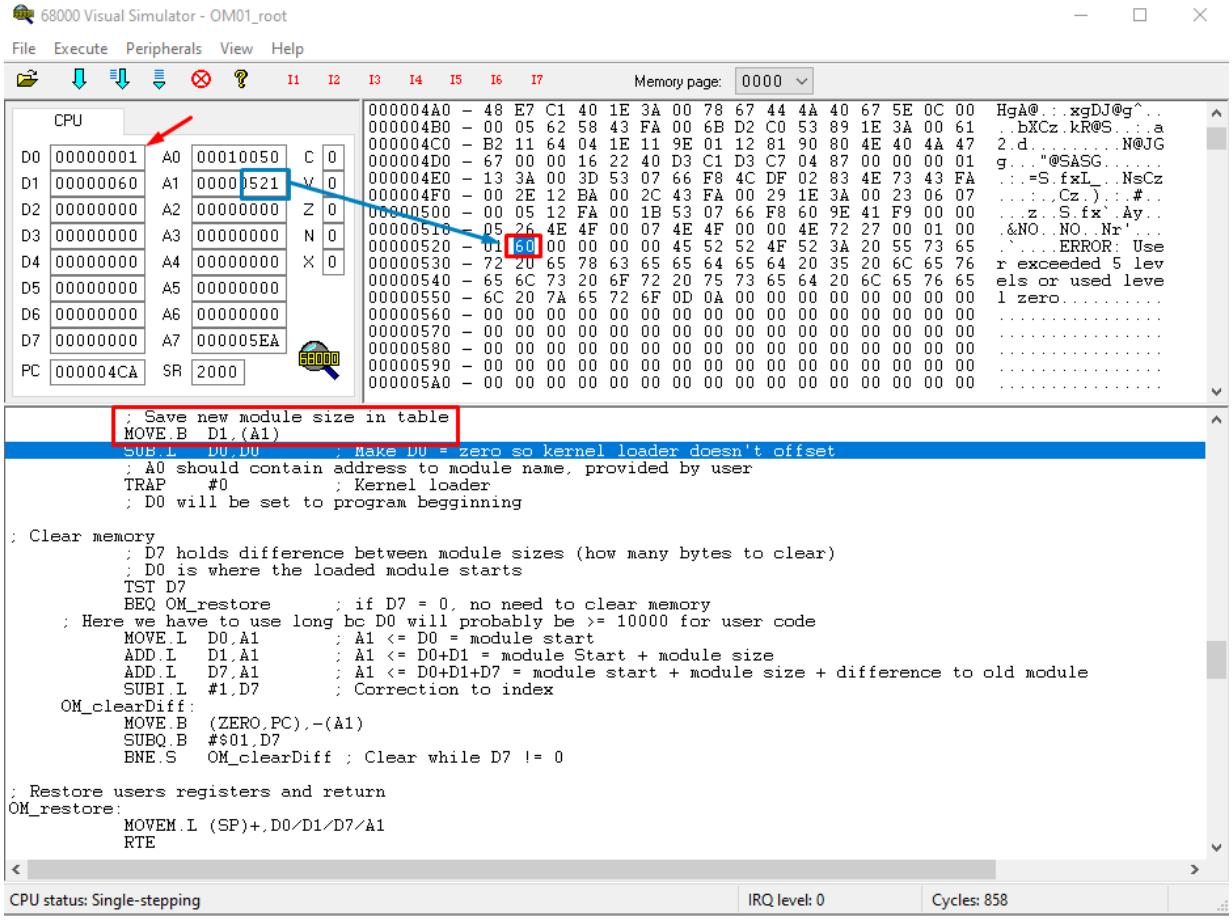

Figura 6: O usuário chamou o overlay manager. Como é sua primeira chamada, ele inicializou a tabela em 0x521 até 0x525 em zero.
Na captura, ele já anotou o tamanho do programa de nível 1 que foi carregado (0x60 = 0d96)

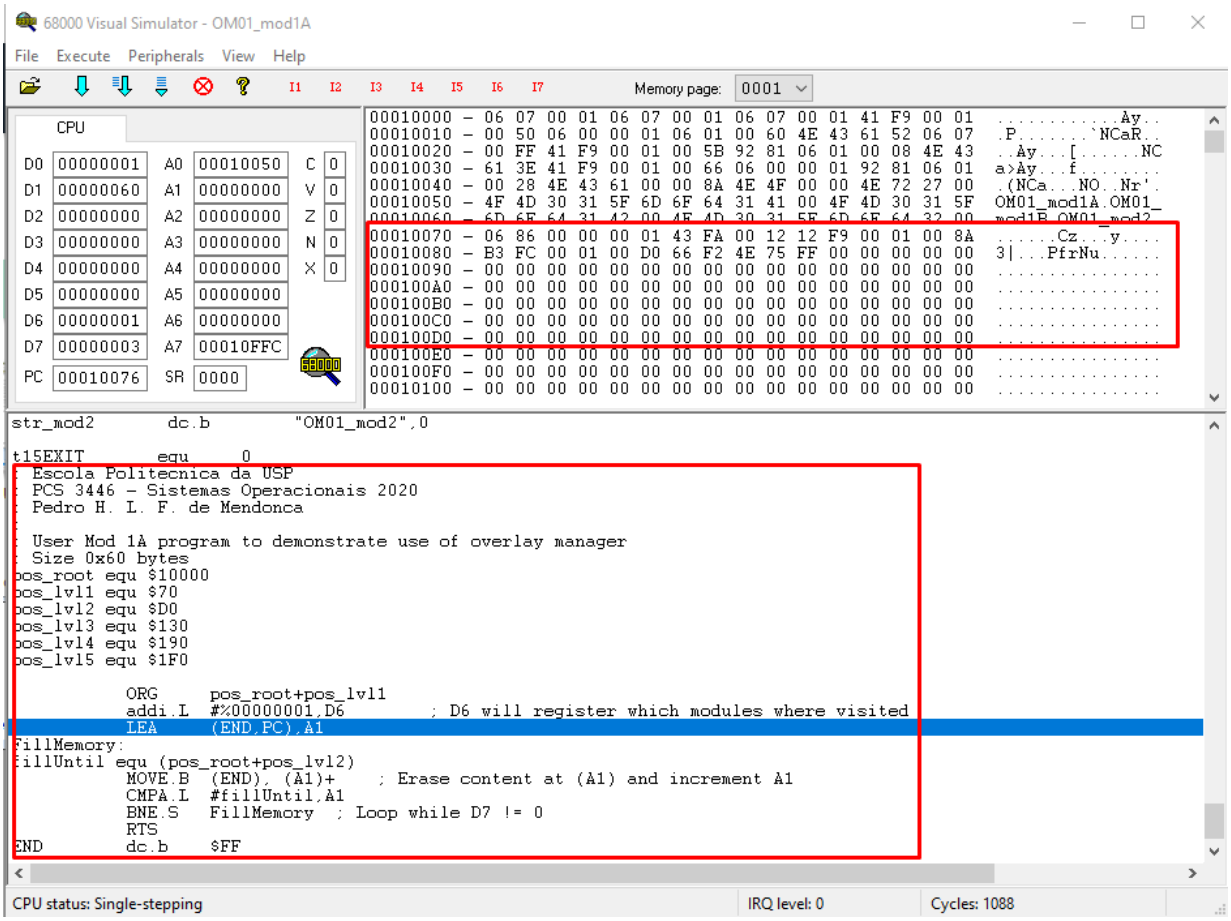

Figura 7: Aqui pode-se ver o módulo 1A (0x10070 até 0x100D0) carregado porém ainda não executado.
Esse código deveria preencher todo conteúdo de memória de 0x1008A até 0x100CF com 0xFF

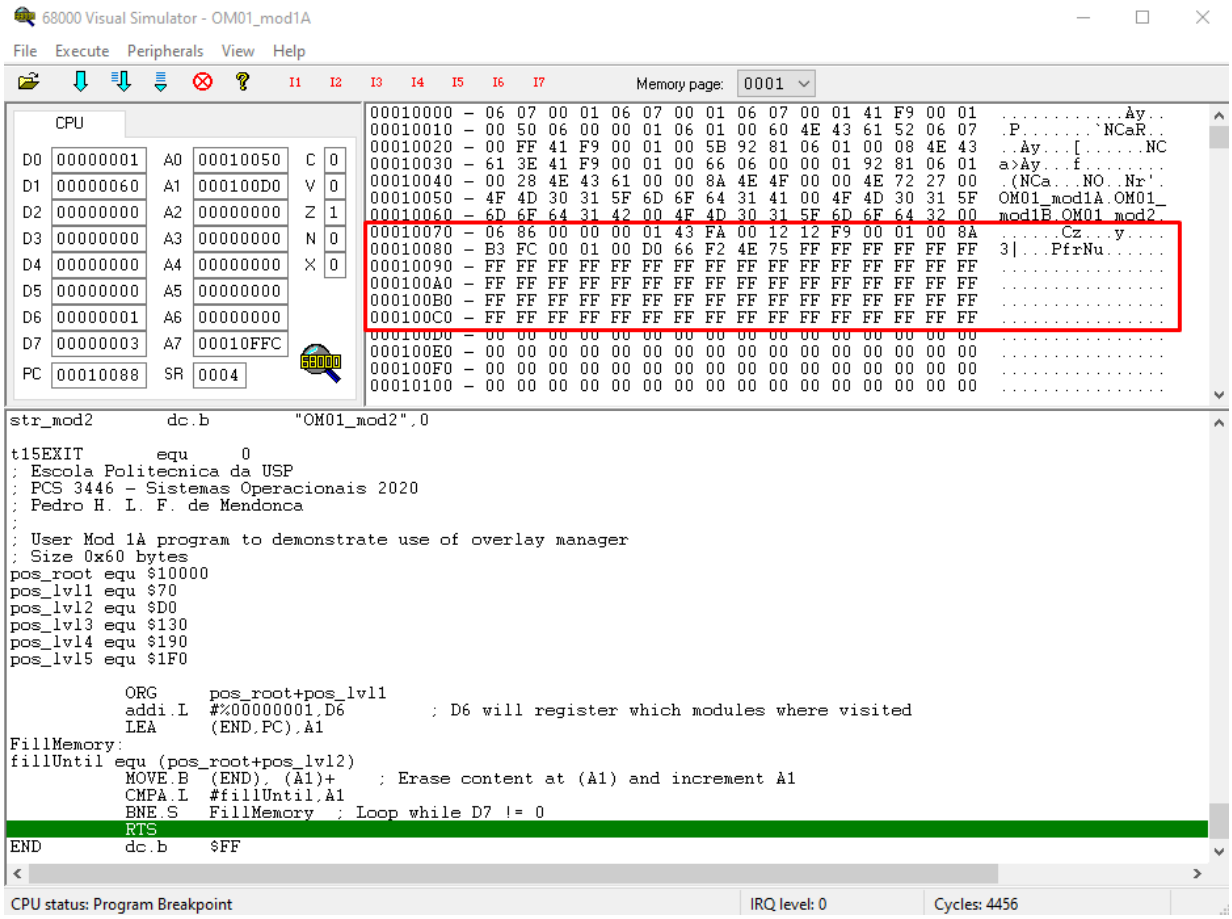

Figura 8: Memória preenchida de 0xFF após execução do módulo 1A. Ao carregar o módulo 1B, de tamanho 0x8 devemos ver o **overlay monitor** apagando a memória usada pelo 1A (que é maior que o 1B)

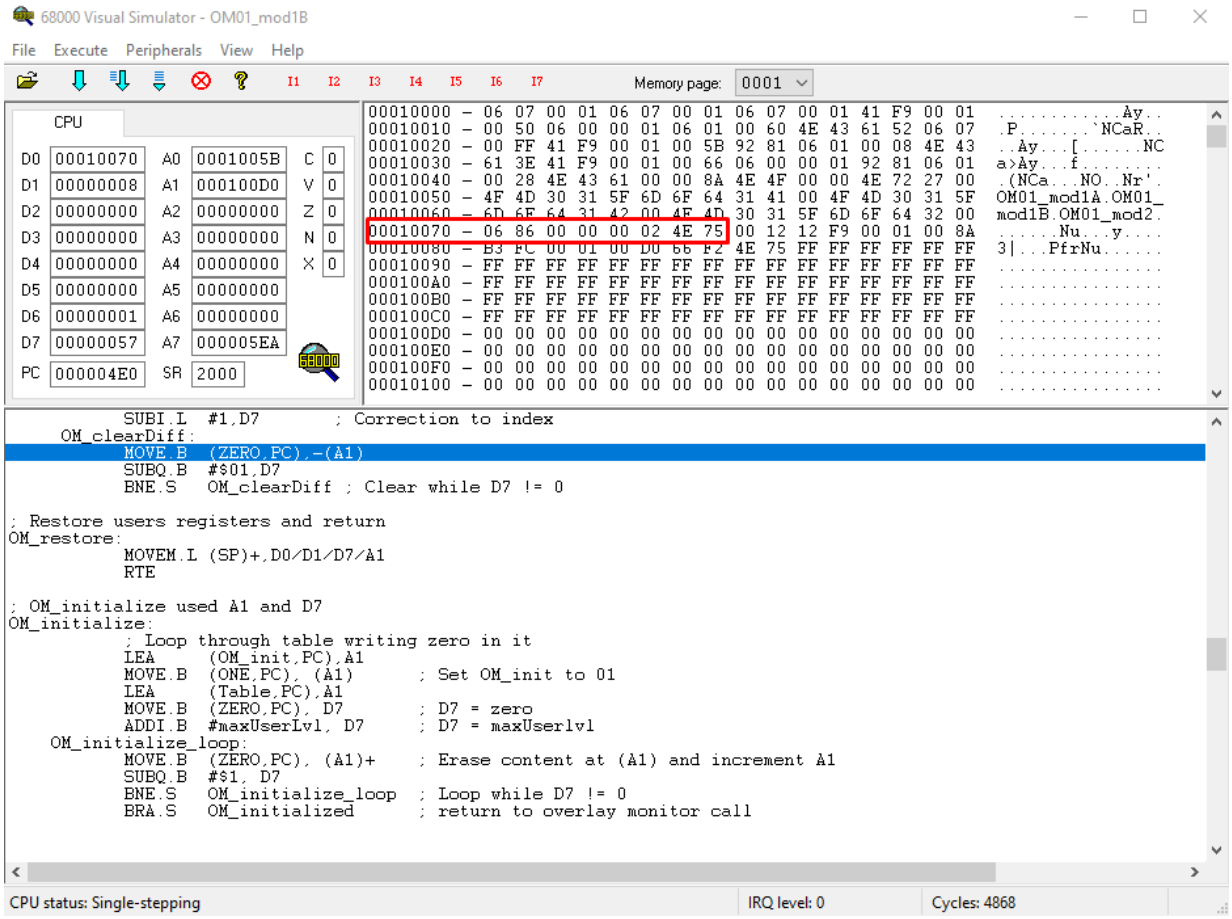

Figura 9: Note que o módulo 1B já fora carregado (de 0x10070 até 0x10078) mas o overlay monitor **ainda** não apagou o resto da memória de nível 1.

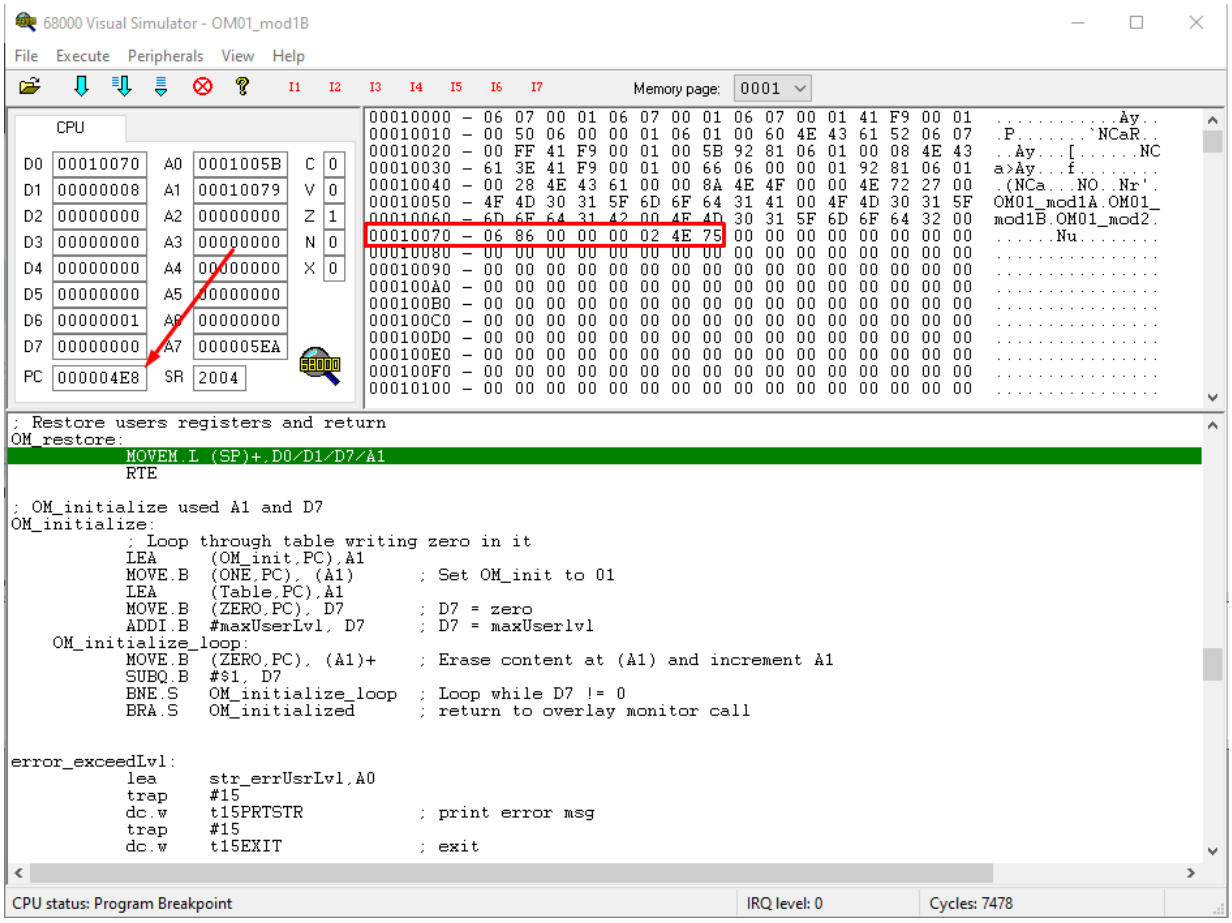

Figura 10: Após o overlay manager apagar a memória remanescente de nível 1. Nessa captura, a execução ainda está dentro do overlay manager (veja o PC de valor baixo e SR com bit 13 (supervisor) settado)

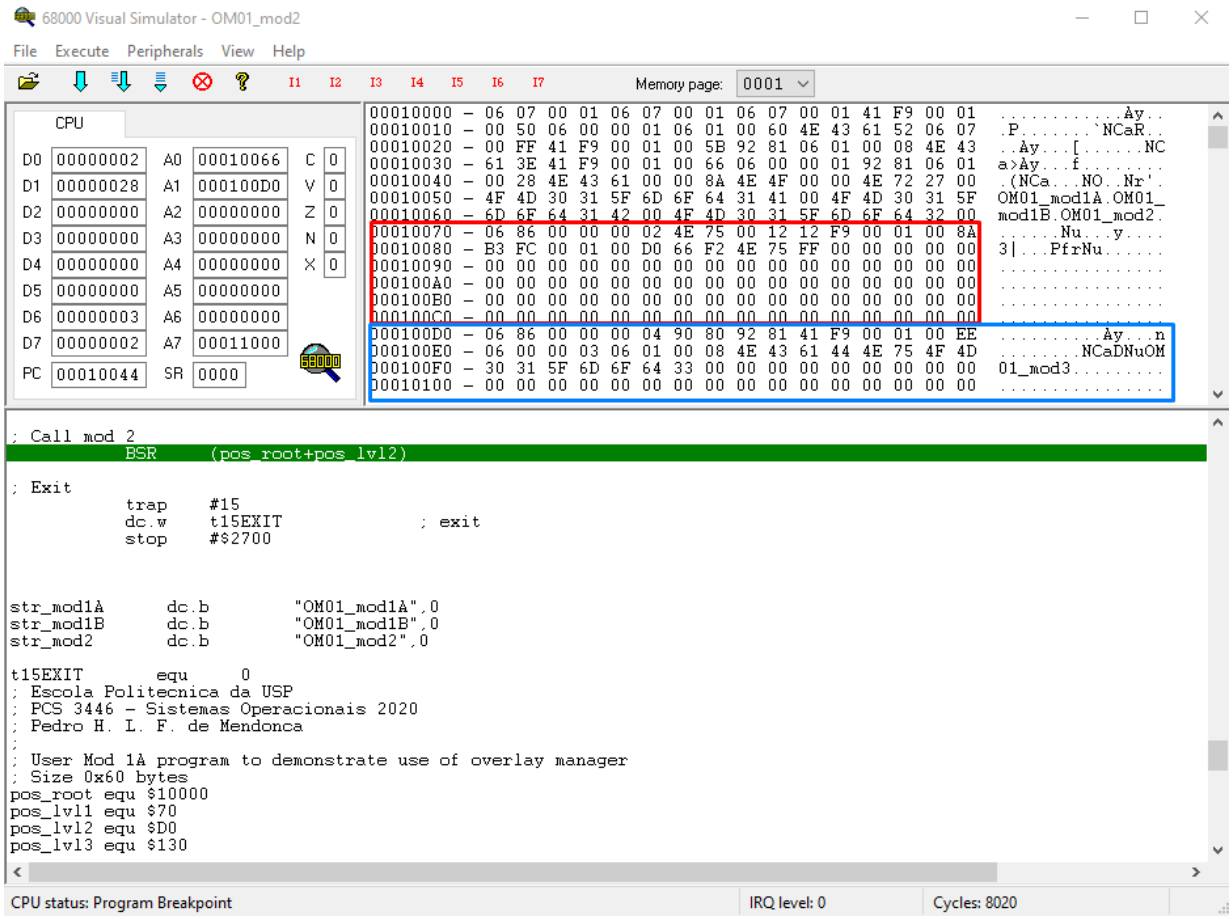

Figura 11: Memória após carregar o nível 2. O root está carregado de 0x10000 a 0x10070, Nível 1 de 0x10070 a 0x100D0 e Nível 2 de 0x100D0 a 0x10130

Note que aloquei 0x60 (0d96) bytes para cada nível, mas os níveis **podem** ter tamanhos diferentes.

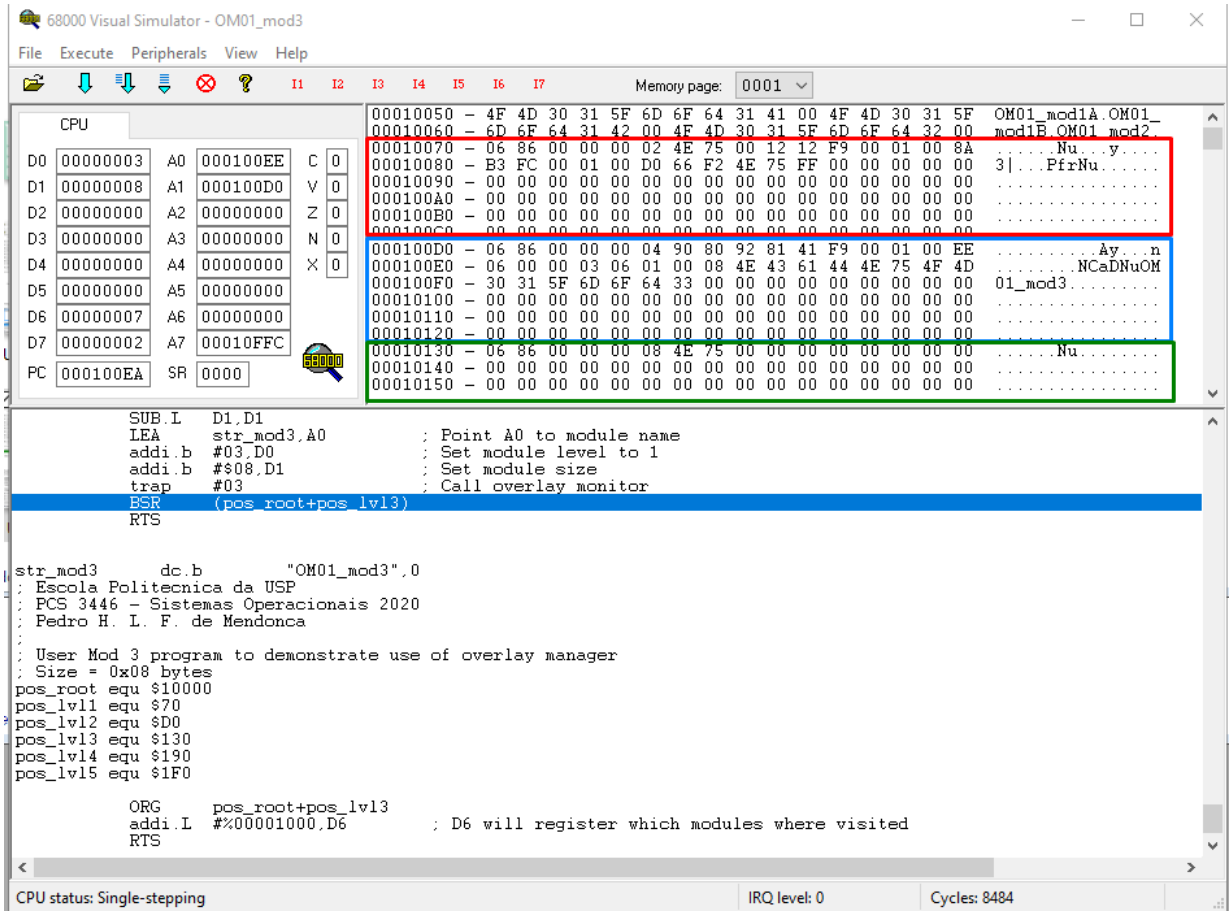

Figura 12: Memória após o nível 2 carregar o nível 3.
O root está carregado de 0x10000 a 0x10070, Nível 1 de 0x10070 a 0x100D0, Nível 2 de 0x100D0 a 0x10130 e Nível 3 de 0x10130 a 0x10190

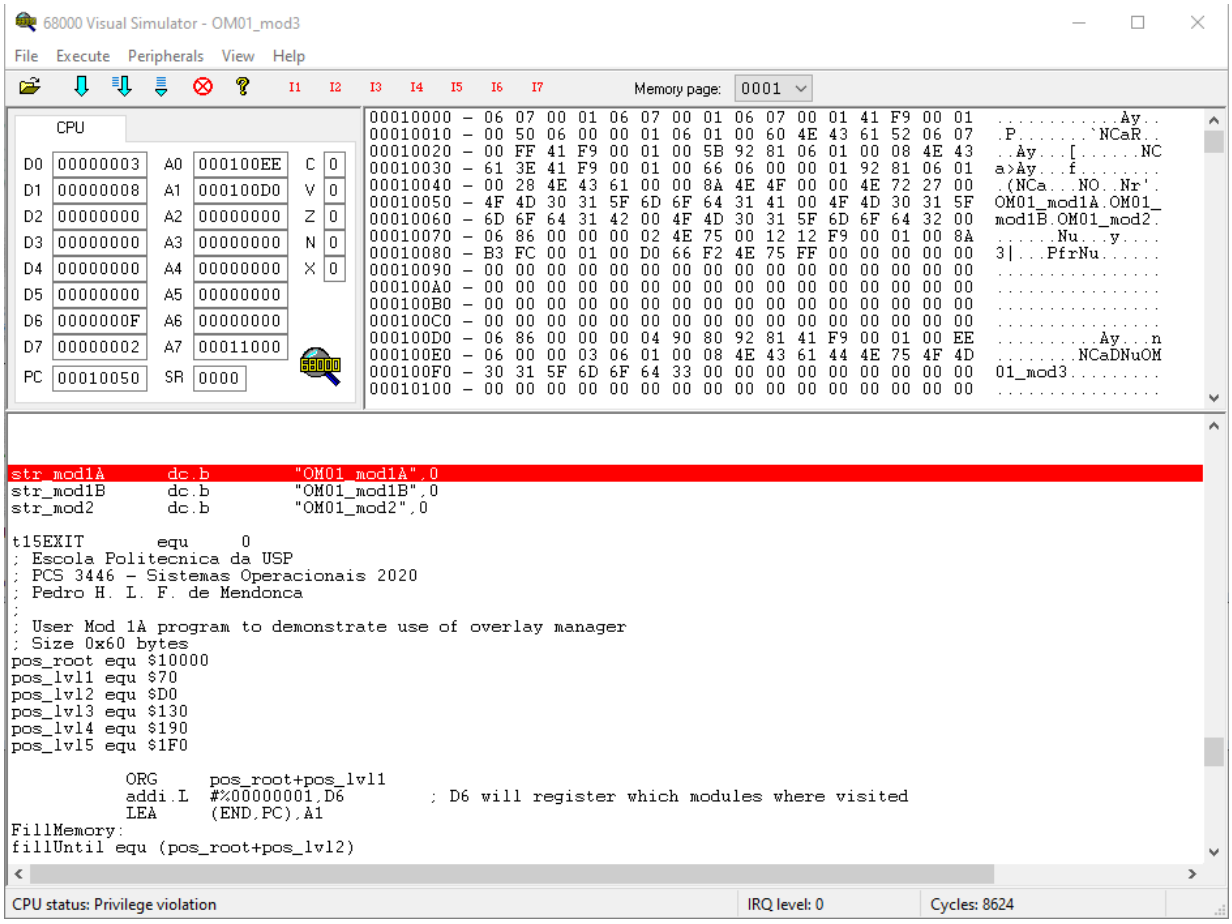

Figura 13: Fim da execução dos módulos e do root. Repare que D6 contém 0xF = 0b1111, logo, todos os módulos executaram.

4 Código Completo

4.1 Kernel

```
1  ; Escola Politecnica da USP
2  ; PCS 3446 - Sistemas Operacionais 2020
3  ; Pedro H. L. F. de Mendonca
4  ;
5  ; Kernel Implementation
6  ;
7  ; Kernel system calls:
8  ;    Trap #0 - Loader (A0 should point to module name as null terminated
   ↪ string and D0 will be overwritten with entry point)
9  ;    Trap #3 - overlay monitor (A0 should point to module name as null
   ↪ terminated string, D0 should be module level and D1 module size)
10 ;
11 ; This kernel loads the overlay monitor (in TRAP #3) and the user root
   ↪ programs
12 ; This implementation uses 44 bytes + overlay monitor
13 ; User program must be at $10000 and is allocated $1000_hex (4096_dec) bytes
   ↪ .
14 ;
15 ;
16
17 ; From 0x000 up to 0x3FF we have interruption tables and simulator data
18 ; So our program starts at 0x400
19             org      $400
20
21 ; TRAP address'es are 0x80 + 4*vector value
22 trap0vec    equ      $80 ; Loader
23 ; TRAP #3 = 0x8C
24 trap3vec    equ      $8C
25
26 ; Set system stack pointer and loader in trap table
27 setup:
28             LEA      sysSP, SP
29             LEA      loader, A0
30             MOVE.L   A0,trap0vec          ; Put address of loader into the trap
                  ↪ vector table #0
31             BRA.S    start
32
33 loader:
34             ; A0 should contain address of module to load (as null terminated
                  ↪ string)
35             ; D0 will be overwritten with loaded modules position in memory
36             TRAP     #15
37             DC.W     t15LOAD              ; load child, entry point in D0
38             BEQ.S    loader_error         ; D0 = 0 on error
```

```
39              RTE
40      loader_error:
41              ; A0 already points to modules name
42              trap    #15
43              dc.w    t15PRTSTR               ; print modulename first
44              lea     errmsg,A0
45              trap    #15
46              dc.w    t15PRTSTR               ; followed by error message
47              trap    #15
48              dc.w    t15EXIT                 ; exit
49              stop    #$2700
50  errmsg  dc.b        ": error loading module",CR,LF,0
51
52  start:
53  ; Load overlay monitor
54              LEA     str_ovrlaymon,A0        ; get name of module to load
55              TRAP    #0                      ; Call loader
56              MOVE.L  D0,trap3vec             ; Put address of overlay monitor into
                    ↪ the trap vector table #3
57              SUB.L   D0,D0                   ; Make D0 = 0 so other loads don't
                    ↪ offset
58
59  ; Load user code
60              LEA     str_userModule,A0       ; get name of module to load
61              TRAP    #0                      ; D0 is loaded with beggining of user
                    ↪ program
62
63  ; Allocate space for user and jump to their code WITHOUT supervisor mode
64              MOVE.L  D0,-(SP)                ; Push D0 (beggining of user space)
                    ↪ in stack
65              MOVE.W  #$0,-(SP)               ; Push user SR in stack
66              ADDI.L  #usrMemSize,D0          ; Allocate usrMemSize for user (D0 =
                    ↪ usr beggining + usr mem size)
67              MOVE.L  D0,A0                   ; Allocate usrMemSize for user (A0 <=
                    ↪ D0)
68              MOVE.L  A0,USP                  ; Allocate usrMemSize for user (User
                    ↪ stack pointer <= A0) (USP <= D0 is invalid)
69              SUB.L   D0,D0                   ; Leave registers in known state for
                    ↪ user
70              SUB.L   A0,A0                   ; Leave registers in known state for
                    ↪ user
71              RTE                             ; Return from Exception (SR <= pop
                    ↪ stack = usr SR, PC <= pop stack = usr program)
72              ; RTE puts SR equal to last stack.W and PC equal to last stack.L
73              ; That's why we pushed 10000 and (SR without supervisor) into
                    ↪ stack.
74
75
76
```

```
77
78 sysSP        equ     $600
79 usrMemSize   equ     $1000               ; User SP points to here initially
80
81
82 ; Strings with null ending
83 str_ovrlaymon       dc.b        "overlay_monitor",0
84
85 str_userModule      dc.b        "OM01_root",0
86 ; ASCII characters
87 LF          equ     $0A
88 CR          equ     $0D
89 ; TRAP #15 Codes
90 t15PRTSTR      equ     7
91 t15EXIT        equ     0
92 t15PRTNUM      equ     5
93 t15GETNUM      equ     6
94 t15LOAD        equ     19
```

4.2 Overlay monitor

```
1  ; Escola Politecnica da USP
2  ; PCS 3446 - Sistemas Operacionais 2020
3  ; Pedro H. L. F. de Mendonca
4  ;
5  ; Overlay monitor implementation
6  ; Overlay monitor has been implemented in such a manner that it is relocable
     ↪ ;
7  ;
8  ;
9  ; Overlay monitor will be called by user using TRAP
10 ; User must put address to module name (as null terminated string) in A0
11 ; D0.BYTE <= Module Level (1 to max)
12 ; D1.BYTE <= Module Size
13 ; User must obey maxUserLvl
14 ; Even using A0, D0 and D1 as parameters, they are NOT edited by the rotine
15 maxUserLvl equ     $5
16 maxUserLvlChar equ '5'
17
18
19             ORG     $4A0 ; Put overlay monitor at ... (it is relocable)
20
21 ; Save user registers
22             MOVEM.L D0/D1/D7/A1,-(SP)
23 ; Check if overlay monitor has been initialized
24             MOVE.B  (OM_init,PC), D7
25             BEQ.S   OM_initialize       ; Goto clear table if not initialized
```

```
26 OM_initialized:
27
28
29 ; Check if module level is compatible (Between 1 and maxUserLvl)
30             TST      D0                    ; Check if D0 == 0
31             BEQ.S    error_exceedLvl
32             CMPI.B   #maxUserLvl, D0
33             BHI.S    error_exceedLvl       ; Check if D0 > maxUserLvl (BHI is
                   ↪ unsigned, BGT is NOT)
34
35 ; Determine entry position in module size table
36             LEA      (Table,PC), A1        ; A1 points to table (index 0)
37             ADD.W    D0,A1                 ; A1 points to user module position
                   ↪ +1
38             SUBQ.L   #1,A1                 ; A1 points to correct user level
                   ↪ table position
39
40              ; A1 holds position to table(module_level)
41 ; If new module size < old module size, calculate memory difference
42             MOVE.B   (ZERO,PC),D7          ; Set memory difference to zero
43             CMP.B    (A1),D1
44             BHS.S    OM_LoadMod            ; size at Table+D0 <= D1, no need to
                   ↪ clear memory (BGE is unsigned, BHS is NOT)
45             MOVE.B   (A1),D7
46             SUB.B    D1,D7                 ; D7 holds size difference of new
                   ↪ module to old module
47              ; A1 holds position to table(module_level)
48              ; D7 holds difference between module sizes (how many bytes to
                   ↪ clear)
49
50 OM_LoadMod:
51 ; Load new module
52              ; Save new module size in table
53             MOVE.B   D1,(A1)
54             SUB.L    D0,D0          ; Make D0 = zero so kernel loader doesn't
                   ↪ offset
55              ; A0 should contain address to module name, provided by user
56             TRAP     #0             ; Kernel loader
57              ; D0 will be set to program begginning
58
59 ; Clear memory
60              ; D7 holds difference between module sizes (how many bytes to
                   ↪ clear)
61              ; D0 is where the loaded module starts
62             TST D7
63             BEQ OM_restore         ; if D7 = 0, no need to clear memory
64       ; Here we have to use long bc D0 will probably be >= 10000 for user
            ↪ code
65             MOVE.L   D0,A1          ; A1 <= D0 = module start
```

```
66              ADD.L   D1,A1           ; A1 <= D0+D1 = module Start + module size
67              ADD.L   D7,A1           ; A1 <= D0+D1+D7 = module start + module size
                    ↪ + difference to old module
68              SUBI.L  #1,D7           ; Correction to index
69          OM_clearDiff:
70              MOVE.B  (ZERO,PC),-(A1)
71              SUBQ.B  #$01,D7
72              BNE.S   OM_clearDiff ; Clear while D7 != 0
73
74  ; Restore users registers and return
75  OM_restore:
76              MOVEM.L (SP)+,D0/D1/D7/A1
77              RTE
78
79  ; OM_initialize used A1 and D7
80  OM_initialize:
81              ; Loop through table writing zero in it
82              LEA     (OM_init,PC),A1
83              MOVE.B  (ONE,PC), (A1)      ; Set OM_init to 01
84              LEA     (Table,PC),A1
85              MOVE.B  (ZERO,PC), D7       ; D7 = zero
86              ADDI.B  #maxUserLvl, D7     ; D7 = maxUserlvl
87          OM_initialize_loop:
88              MOVE.B  (ZERO,PC), (A1)+    ; Erase content at (A1) and increment
                    ↪ A1
89              SUBQ.B  #$1, D7
90              BNE.S   OM_initialize_loop  ; Loop while D7 != 0
91              BRA.S   OM_initialized      ; return to overlay monitor call
92
93
94  error_exceedLvl:
95              lea     str_errUsrLvl,A0
96              trap    #15
97              dc.w    t15PRTSTR           ; print error msg
98              trap    #15
99              dc.w    t15EXIT             ; exit
100             stop    #$2700
101
102 OM_init     ds.b    1
103 ZERO        dc.b    $00
104 ONE         dc.b    $01
105
106 ; This table stores each levels max size
107 Table       ds.b    maxUserLvl
108
109 ; Strings with null ending
110 str_errUsrLvl       dc.b        "ERROR: User exceeded ",maxUserLvlChar, "
    ↪ levels or used level zero",CR,LF,0
111
```

```
112 ; ASCII characters
113 LF          equ     $0A
114 CR          equ     $0D
115
116 ; TRAP #15 Codes
117 t15PRTSTR       equ     7
118 t15EXIT         equ     0
119 t15PRTNUM       equ     5
120 t15GETNUM       equ     6
121 t15LOAD         equ     19
```

4.3 User Root

```
1  ; Escola Politecnica da USP
2  ; PCS 3446 - Sistemas Operacionais 2020
3  ; Pedro H. L. F. de Mendonca
4  ;
5  ; User root program to demonstrate use of overlay manager
6  ; Size 0x66
7  pos_root equ $10000
8  pos_lvl1 equ $70
9  pos_lvl2 equ $D0
10 pos_lvl3 equ $130
11 pos_lvl4 equ $190
12 pos_lvl5 equ $1F0
13
14
15             ORG     pos_root
16 ; Do work
17             addi.b  #01,D7
18             addi.b  #01,D7
19             addi.b  #01,D7
20
21 ; Load Module Level 1 A
22             LEA     str_mod1A,A0        ; Point A0 to module name
23             addi.b  #01,D0              ; Set module level to 1
24             addi.b  #(pos_lvl2-pos_lvl1),D1     ; Set module size
25             trap    #03                 ; Call overlay monitor
26
27 ; Call mod 1
28             BSR     (pos_root+pos_lvl1)
29
30 ; Return from mod 1
31             addi.b #$FF,D7
32
33 ; Load Module Level 1 B
34             LEA     str_mod1B,A0 ; Point A0 to module name
```

```
35              SUB.L   D1,D1
36              addi.b  #$08,D1         ; Set module size
37              trap    #03             ; Call overlay monitor
38
39  ; Call mod 1
40              BSR     (pos_root+pos_lvl1)
41
42  ; Load Module Level 2
43              LEA     str_mod2,A0 ; Point A0 to module name
44              addi.b  #01,D0          ; Set module level = 2
45              SUB.L   D1,D1
46              addi.b  #$28,D1         ; Set module size
47              trap    #03             ; Call overlay monitor
48
49  ; Call mod 2
50              BSR     (pos_root+pos_lvl2)
51
52  ; Exit
53              trap    #15
54              dc.w    t15EXIT                 ; exit
55              stop    #$2700
56
57
58
59  str_mod1A       dc.b        "OM01_mod1A",0
60  str_mod1B       dc.b        "OM01_mod1B",0
61  str_mod2        dc.b        "OM01_mod2",0
62
63  t15EXIT     equ     0
```

4.4 User level 1A

```
1   ; Escola Politecnica da USP
2   ; PCS 3446 - Sistemas Operacionais 2020
3   ; Pedro H. L. F. de Mendonca
4   ;
5   ; User Mod 1A program to demonstrate use of overlay manager
6   ; Size 0x60 bytes
7   pos_root equ $10000
8   pos_lvl1 equ $70
9   pos_lvl2 equ $D0
10  pos_lvl3 equ $130
11  pos_lvl4 equ $190
12  pos_lvl5 equ $1F0
13
14              ORG     pos_root+pos_lvl1
```

```
15              addi.L   #%00000001,D6          ; D6 will register which modules
                    ↪ where visited
16              LEA      (END,PC),A1
17  FillMemory:
18  fillUntil equ (pos_root+pos_lvl2)
19              MOVE.B   (END), (A1)+     ; Erase content at (A1) and increment A1
20              CMPA.L   #fillUntil,A1
21              BNE.S    FillMemory   ; Loop while D7 != 0
22              RTS
23  END         dc.b     $FF
```

4.5 User Level 1B

```
1  ; Escola Politecnica da USP
2  ; PCS 3446 - Sistemas Operacionais 2020
3  ; Pedro H. L. F. de Mendonca
4  ;
5  ; User Mod 1B program to demonstrate use of overlay manager
6  ; Size = 0x08 bytes
7  pos_root equ $10000
8  pos_lvl1 equ $70
9  pos_lvl2 equ $D0
10 pos_lvl3 equ $130
11 pos_lvl4 equ $190
12 pos_lvl5 equ $1F0
13
14              ORG      pos_root+pos_lvl1
15              addi.L   #%00000010,D6          ; D6 will register which modules
                    ↪ where visited
16              RTS
```

4.6 User Level 2

```
1  ; Escola Politecnica da USP
2  ; PCS 3446 - Sistemas Operacionais 2020
3  ; Pedro H. L. F. de Mendonca
4  ;
5  ; User Mod 2 program to demonstrate use of overlay manager
6  ; Size = 0x28 bytes
7  pos_root equ $10000
8  pos_lvl1 equ $70
9  pos_lvl2 equ $D0
10 pos_lvl3 equ $130
11 pos_lvl4 equ $190
12 pos_lvl5 equ $1F0
```

```
13
14              ORG     pos_root+pos_lvl2
15              addi.L  #%00000100,D6          ; D6 will register which modules
                    ↪ where visited
16
17              ; Load Level 3
18              SUB.L   D0,D0
19              SUB.L   D1,D1
20              LEA     str_mod3,A0            ; Point A0 to module name
21              addi.b  #03,D0                 ; Set module level to 1
22              addi.b  #$08,D1                ; Set module size
23              trap    #03                    ; Call overlay monitor
24              BSR     (pos_root+pos_lvl3)
25              RTS
26
27
28 str_mod3      dc.b        "OM01_mod3",0
```

4.7 User Level 3

```
1  ; Escola Politecnica da USP
2  ; PCS 3446 - Sistemas Operacionais 2020
3  ; Pedro H. L. F. de Mendonca
4  ;
5  ; User Mod 3 program to demonstrate use of overlay manager
6  ; Size = 0x08 bytes
7  pos_root equ $10000
8  pos_lvl1 equ $70
9  pos_lvl2 equ $D0
10 pos_lvl3 equ $130
11 pos_lvl4 equ $190
12 pos_lvl5 equ $1F0
13
14              ORG     pos_root+pos_lvl3
15              addi.L  #%00001000,D6          ; D6 will register which modules
                    ↪ where visited
16              RTS
```