|  |  |  |  |
| --- | --- | --- | --- |
| **Conjunto de Instruções uJVM** | | | |
| **Código** | **Instrução** | **Pilha (estack)** | **Semântica** |
| 1 | **load** b | …  …, val | push(local[FP+b]); |
| 2 | **store** b | …, val  … | local[FP+b] = pop(); |
| 3 | **getstatic** s | …  …, val | push(data[s]); |
| 4 | **putstatic** s | …, val  … | data[s] = pop(); |
| 5 | **const** w | …  …, val | push(w); |
| 6 | **add** | …, v1, v2  …, v1+v2 | push(pop() + pop()); |
| 7 | **sub** | …, v1, v2  …, v1-v2 | push(- pop() + pop()); |
| 8 | **mul** | …, v1, v2  …, v1\*v2 | push(pop() \* pop()); |
| 9 | **div** | …, v1, v2  …, v1/v2 | x = pop();  se x != 0 push(pop() / x) senão erro; |
| 10 | **rem** | …, v1, v2  …, v1%v2 | x = pop();  se x != 0 push(pop() % x) senão erro; |
| 11 | **neg** | …, v1  …, - v1 | push(- pop()); |
| 12 | **newarray** | …, n  …, adr | n = pop();  reserve vetor de n words a partir de adr;  inicie posições do vetor com 0;  push(adr); |
| 13 | **aload** | …, adr, i  …, val | i = pop(); adr = pop();  push(heap[adr + i + 1]); |
| 14 | **astore** | …, adr, i, val  … | val = pop(); i = pop(); adr = pop();  heap[adr + i + 1] = val; |
| 15 | **arraylength** | …, adr  …, len | adr = pop();  push(heap[adr]); |
| 16 | **pop** | …, val  … | x = pop(); |
| 17 | **jmp** s |  | PC = s; |
| 18 a 23 | **j***c* s  *c*=**eq**|**ne**|**lt**|**le**|**gt**|**ge** | …, x, y  … | y = pop(); x = pop();  se (x *c* y) PC = s; |
| 24 | **call** s |  | PPUSH(PC+3);  PC = s; |
| 25 | **return** |  | PC = PPOP(); |
| 27 | **enter** b1 b2 |  | pnum = b1; lnum = b2;  // salva contexto  PPUSH(FP); FP = SP; SP = SP + lnum;  inicie frame para 0;  // copia parametros  for (i=pnum-1; i>=0; i--)  local[FP+i] = pop(); |
| 28 | **exit** |  | SP = FP; FP = PPOP(); |
| 29 | **printi** | …, val  … | imprime pop(); |
| 30 | **scani** | …  …, val | val = leia\_inteiro\_teclado() ou 0 se erro  push(val); |
| 31 | **prints** s |  | for (i=s; code[i] != 0; i++)  imprime code[i] |
| 32 | **trap** b |  | Imprime msg de erro e para VM;  se b = 1, msg = “função sem return”; |
| b = byte (8 bits), s = short (16), w = word (32) | | | |

|  |  |
| --- | --- |
| **Formato de código OBJ** | |
| **Bytes** | **Conteúdo** |
| 2 | UP |
| 4 | Tamanho do código + strzs (*n*+*m*) |
| 4 | Número de words em data |
| 4 | Valor do mainPC = início de bloco principal |
| 4 | Valor do strzStart = início de área de strings |
| *n* | Área de código |
| *m* | Área de strings (constantes string) |

|  |  |
| --- | --- |
| **Registradores & Memórias** | |
| **Nome** | **Conteúdo** |
| code | Código (byte) |
| data | Globais (word) |
| heap | Vetores (word) |
| pstack | Pilha de Frames (word)  Acessada como pilha (via PPOP() e PPUSH()) ou vetor (local[]) |
| estack | Pilha de Expressão (word)  Acessada como pilha via pop() e push() |
| PC | Program Counter  Instrução a executar em code |
| FP | Frame Pointer  Início da área local em estack |
| SP | Stack Pointer  Início da área livre em estack |

|  |  |
| --- | --- |
| **Exemplo de Código em uC** | |
| /\* exemplo \*/  **int** a;  **int** sumA(**int** p1)  **{**  return p1 + a;  **}**  **void** main()  **{**  a = 4;  **printf**("3+a: ", sumA(3), "\n");  **}** | |
| **Adr** | **Código uJVM** |
| 0  3  5  8  9  10  11  14  19  22  25  30  33  34  37  38  39  45 | enter 1 1  load 0  getstatic 0  add  exit  return  enter 0 0 // mainPC  const 4  putstatic 0  prints 39  const 3  call 0  printi  prints 45  exit  return  3+a: \0 // strzStart  \n\0 |