1. ① keyboard 产生 Value ② keyboard 至状态位为 1 ③ Processor 收到状态位 ④ Processor 读 Value 并至状态位为 0 (不需要代码体现)隐式

keyboard status register

KBSR 高8位 [15:8] 为0，低8位为 8-bit ASCII | KBSR 最高位第15位为 Status bit，1: ready 0: not ready

KBDR: keyboard data register

2. Display: ① Processor 写入 DDR，隐式至状态位为0 ② Display 读 DDR 输出 ③ Display 至状态位为1 ④ Processor 警惕状态位变1后继续写下个字符

DDR 高位 [15:8] 为0，低位 [7:0] 为 2EXT8-bit ASCII. | DSR 最高位 [15] 为 Status bit 其余位均不用 → 1000…… ready 0000…0 not ready

3. LDI 与 LD 注意区别. LDI R4, DSR (DSR中为地址) | LD R3, ZERO (ZERO中为值) | STI R0, DDR

DSR . FILL xFE04 查看 xFE04 地址内值 DDR . FILL xFE06 写入 xFE06 对应地址内

4. JMP BaseR; PC ← BaseR | JSR: R7 ← PC, PC ← PC + SEXT16 (PCoffset11) | JSRR: R7 ← PC, PC ← BaseR

5. B为某一Regisister，若子程序运行后B改变，则为 Caller-Saved，若B不改变，则为 Callee-Saved 子程序内部 ST. LD3.

| R7 和 any output register 都为 Caller-Saved

6. Subroutine Calling Interface: ① Subroutine input ② Subroutine Output ③ Ownership of other registers ④ Side effect

7. PSR=0 (privileged), PSR=1 (unprivileged)

8. TRAP: R7 ← PC, PC ← M[ZEXT16(vec8)], x0000 - x00FF 中存的是 TRAP 子程序的起始地址 (叫做 Trap Vector Table)

9. Stack: Push: ADD R6, R6, #-1 / STR R?, R6, #0 | Pop: ADD R6, R6, #1

10. 栈帧 Stack frame: ① 局部变量 ② Caller Stack Frame 的地址 ③ 返回地址 ④ Output ⑤ Input

11. C 的 IO: "\\" 代表一个反斜杠 | printf: %e: double as decimal scientific notation, %f(%.9f): double as decimal, %x: 小写16进制, %X: 大写16进制, %%: 打印一个百分号, %u: unsigned int as decimal

Scanf: %f: convert decimal real number to float, %lf: convert decimal real number to double, %u: unsigned int %x or %X: 16进制转为 unsigned int

Scanf 变量前要加 & | printf 返回被打印的字符数，-1 on error, scanf 返回 the number of conversions performed successfully, or -1 for no conversions (-1: 无数据)

12. 除法 "/": round towards 0. A%B 定义为: (A/B)*B + (A%B) is equal to A.

取模 "%" 和除法 "/" 都是先当正数算，然后考虑符号，Eg. (-11%3) 是 -2

13. 右移 >> { 2's complement: 算数右移, unsigned: 逻辑右移 }, 左移 <<: 都是补0

右移均为下取整: $A >> N = \lfloor \frac{A}{2^N} \rfloor$, $-120 >> 4 = \lfloor \frac{-120}{2^4} \rfloor = -8$, $120 >> 4 = \lfloor \frac{120}{2^4} \rfloor = 7$

14. 赋值号左侧必有明确的地址 (错误例子: A+B=42)

A=B=0; 与 A=(B=0); 等价. A=x 这个表达式本身的返回值为 "=" 右侧的值，所以(B=0)返回0

15. Be careful with Auto-Conversion!

16. && 和 || 都是短路的，-=, *=, /=, %=, |=, ^=, &=, <<=, >>= 都可以用

17. i++: read the value, then increment i

++i: increment i, then read it

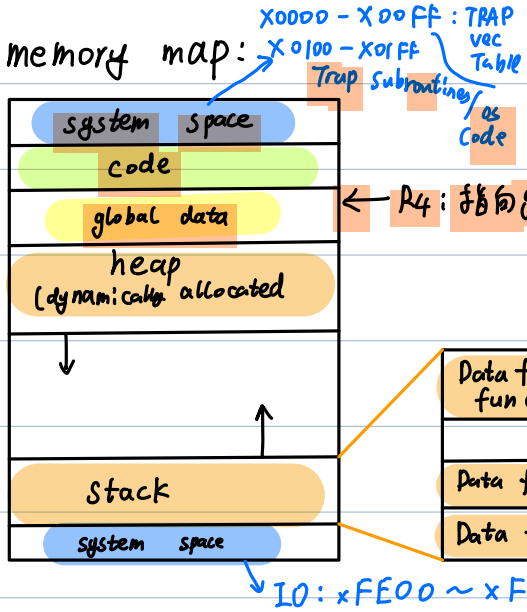
18. C 的 Storage Class: Static (全局) (global data), automatic (局部) (stack), dynamic (临时) (heap, 堆) ← Must be tracked by program

Static 在函数外全局变量 作用域改变 { 不加 static: Global scope; 加 static: file scope }

Static 在函数内改变存储类 (storage class) 但作用域与局部 function/block scope { 不加 static: automatic (stack); 加 static: static (global data) }

总结: 带 "static" 的均为 Static (global data) 存在 global data 区域

19. memory map:



21、全局变量可永限自动初始化为0，局部变量无法自动初始化

22、LC-3 Stack frame: (栈帧底部)
⑤ Local variable
④ Address of Caller's stack frame
③ Return address (R7 in LC-3)
② Outputs (Return value)
① Inputs (parameters, arguments) (栈帧底部)

Caller负责将函数参数压入栈(①), Callee负责②~⑤

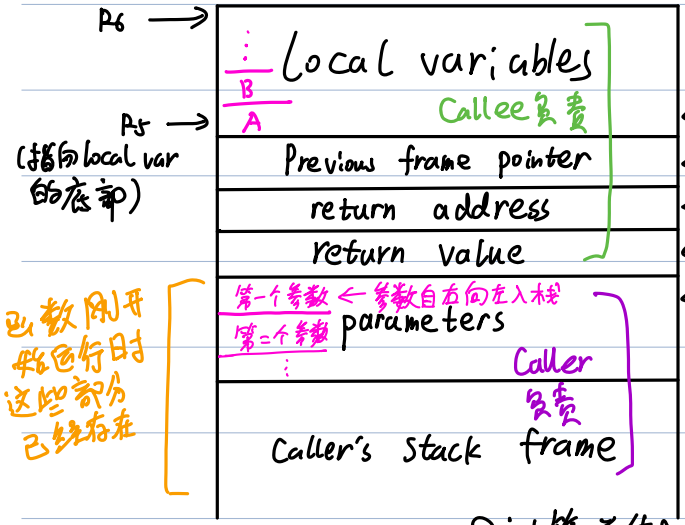
☆2

返回时，③~⑤ Pop出栈，

Return value 别在栈顶

Caller可用R6直接访问返回值

23. Stack Map



返回前弹出，栈顶为返回值

☆

(参数从右向前push入栈)

Note: 全局变量通过R4调用(R4+0, R4+1, …)
局部变量通过R5调用(R5-0, R5-1, …)

24、调用函数步骤：①计算并传入参数(将参数压入栈)②调用函数③从栈顶读取返回值④返回值和传入参数出栈

25、函数代码4部分：①设置栈帧②运行程序③拆除栈帧④返回return

栈帧设定方法：
```
ADD R6, R6, #-4 → make space for the remainder of the stack frame
STR R5, R6, #1  → save Caller's frame pointer into stack frame
ADD R5, R6, #0  → set frame pointer
STR R7, R5, #2  → save return address into stack frame
```

返回值存储：
```
LDR R0, R5, #?
STR R0, R5, #3
```

返回步骤：
```
LDR R7, R5, #2 → restore return address from the stack frame
LDR R5, R5, #1 → restore Caller's frame pointer from the stack frame
ADD R6, R6, #3 → Pop down to return value
RET
```

26、如果一个子程序本身还调用其它子程序，则必须要保存R7 ☆

27、Static变量仅初始化一次，之后保留上一次运算后的值 ☆

28.

A	B	C	D	E	F
10	11	12	13	14	15
1010	1011	1100	1101	1110	1111

别漏分号在语句结束!!!

☆

(NOT A = xFFFF - A , -A = x10000 - A = xFFFF - A + 1)