Stack | Heap data	Activation records: local variables, parameters, etc. Dynamically allocated data—new or malloc()
Static data	Global data and static local data
Text	Machine code instructions (and some constants)
Reserved	Reserved for operating system

Symbol & Relocation Table

LD - load	Text - this file's instructions
ST - store	Data - this file's .fills
BL - branch	Unknown - Not in this file

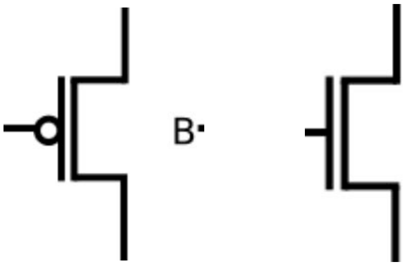
Reloc注意: 填写的是全局变量，函数内部变量不要填。Symbol一定要注意printf这种基础的函数也要写进去。

Object code format

- Header
- Text
- Data
- Symbol table
- Relocation table (maps symbols to instructions)
- Debug info

Transistor

左: 通电不导通　　右: 通电导通

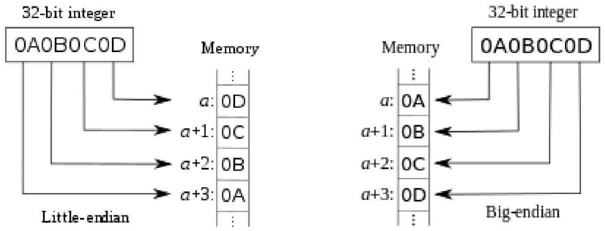


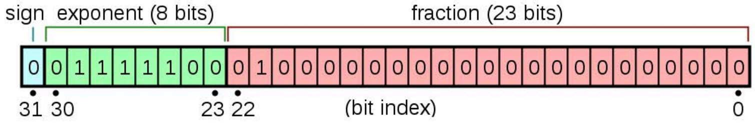
Tiancheng Jiao
EECS 370
tcjiao

	LC2K	LEG
# registers	8	32
Register width	32 bits	64 bits
Memory size	2^{18} bytes	2^{64} bytes
# instructions	8	40-ish
Addressability	Word	Byte

Big Endian vs. Little Endian

- Endian-ness: ordering of bytes within a word
  - Little - increasing numeric significance with increasing memory addresses
  - Big – The opposite, most significant byte first
  - The Internet is big endian, x86 is little endian, LEG and ARMv8 can switch
    - But in general assume little endian. (Figures from Wikipedia)

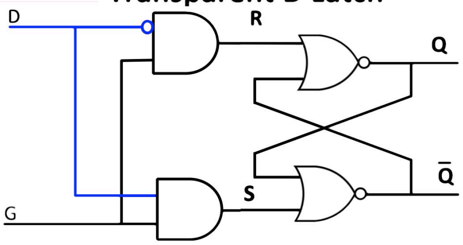




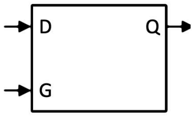
$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\ldots b_{23})_2-127} \times (1.b_{22}b_{21}\ldots b_0)_2,$$

0000	EQ	Equal	Z==1
0001	NE	Not equal	Z==0
1010	GE	Signed greater than or equal	N==V
1011	LT	Signed less than	N!=V
1100	GT	Signed greater than	Z==0 && N==V
1101	LE	Signed less than or equal	!(Z==0 && N==V)
1110	AL	Always	Any
1111	NV		

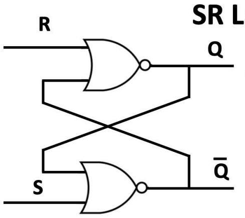
Transparent D Latch



D	G	Q	Q̄
0	0	Q	Q̄
0	1	0	1
1	0	Q	Q̄
1	1	1	0

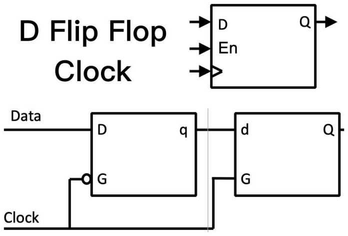


SR Latch

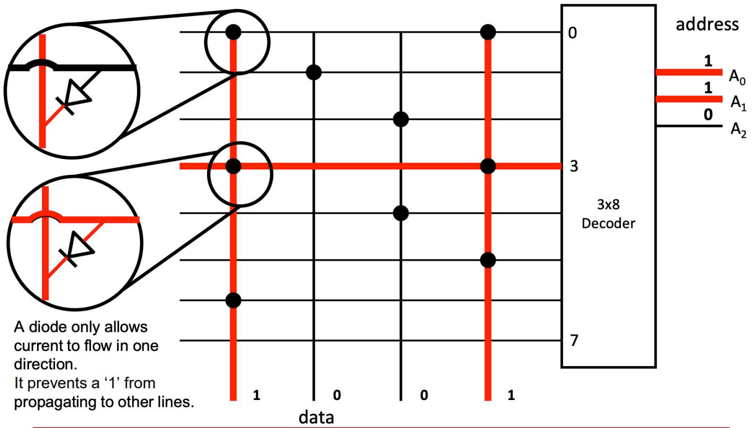


S	R	Q	Q̄
0	0	Q	Q̄
0	1	0	1
1	0	1	0
1	1	BAD	

D Flip Flop



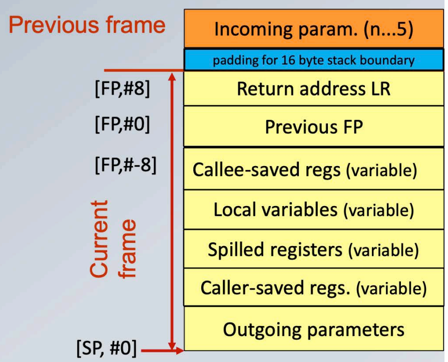
8-entry 4-bit ROM



A diode only allows current to flow in one direction.
It prevents a '1' from propagating to other lines.

LEGv8 Stack Frame

- ❑ Must be aligned on 16 byte boundaries
- ❑ FP (Frame pointer, found in X29) provides a fixed address from which to access items in the current stack frame
- ❑ Stack frames are connected via a linked list of FPs
- ❑ We can do without frame pointers if we carefully track the stack pointer (SP), **but** FP makes life easier…

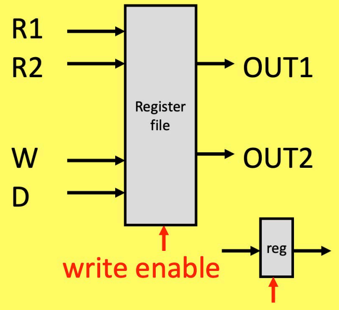


Previous frame

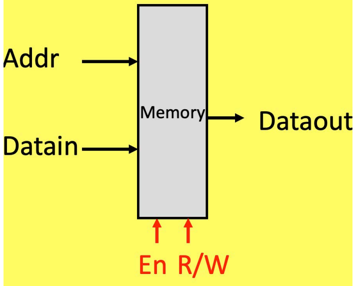
Incoming param. (n…5)
padding for 16 byte stack boundary
[FP,#8] Return address LR
[FP,#0] Previous FP
[FP,#-8] Callee-saved regs (variable)
Local variables (variable)
Spilled registers (variable)
Caller-saved regs. (variable)
[SP, #0] Outgoing parameters

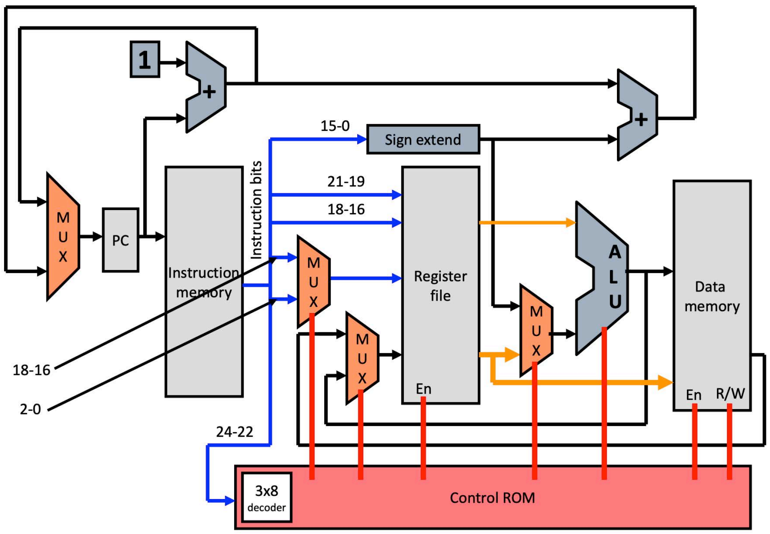
Current frame

Register File or Register

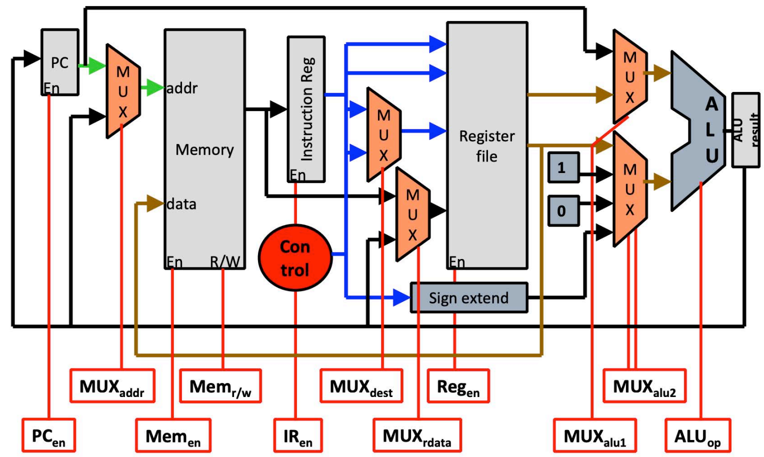


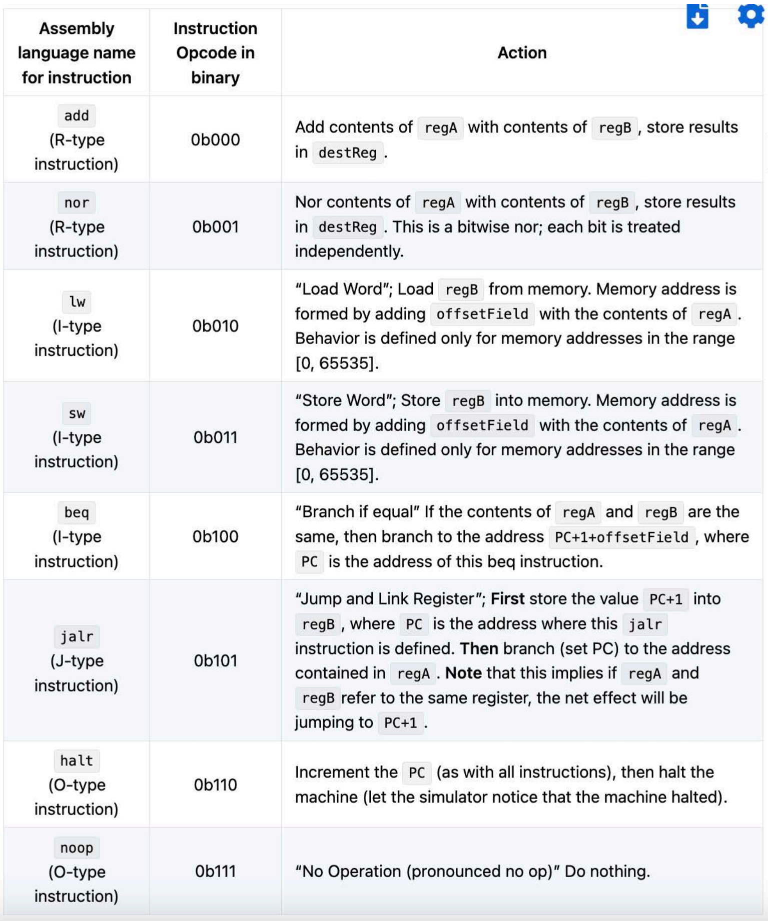
Memory



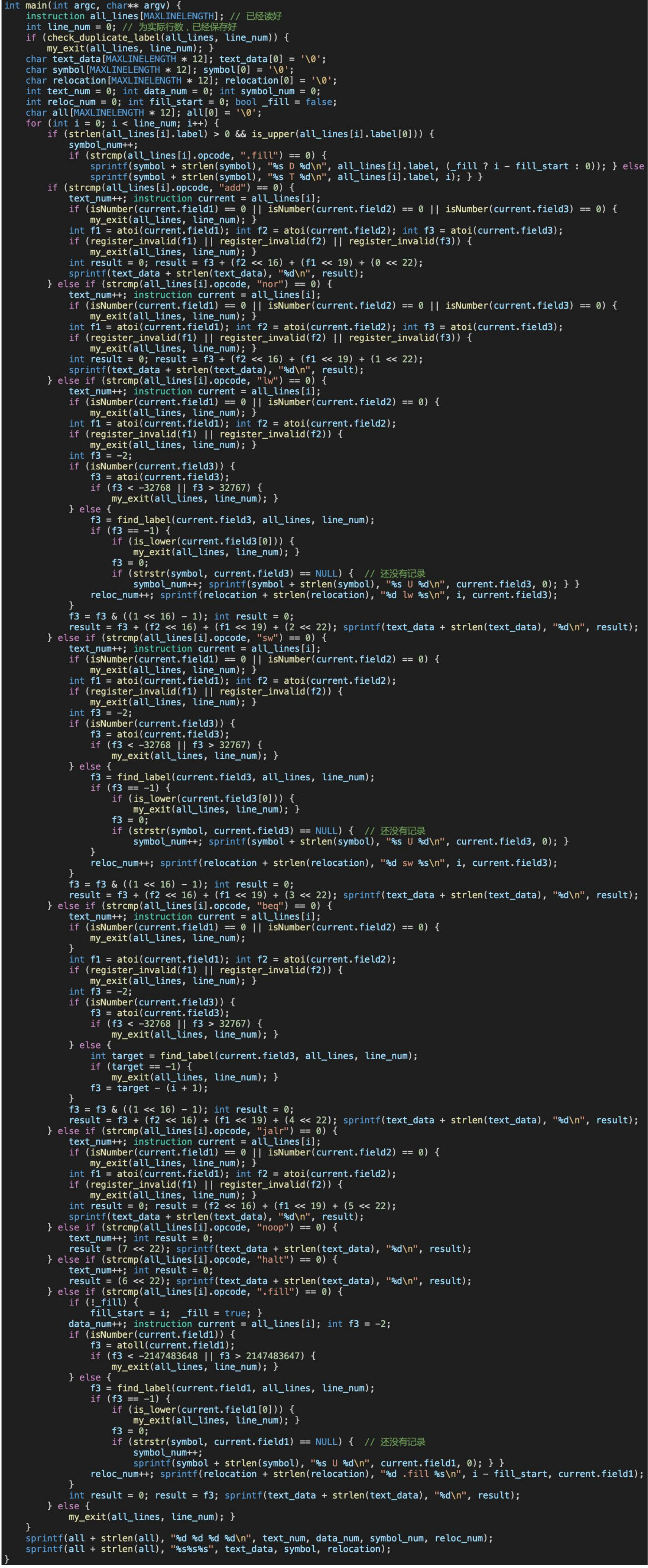


Multicycle LC2K Datapath





Assembly language name for instruction	Instruction Opcode in binary	Action
`add` (R-type instruction)	0b000	Add contents of `regA` with contents of `regB`, store results in `destReg`.
`nor` (R-type instruction)	0b001	Nor contents of `regA` with contents of `regB`, store results in `destReg`. This is a bitwise nor; each bit is treated independently.
`lw` (I-type instruction)	0b010	"Load Word"; Load `regB` from memory. Memory address is formed by adding `offsetField` with the contents of `regA`. Behavior is defined only for memory addresses in the range [0, 65535].
`sw` (I-type instruction)	0b011	"Store Word"; Store `regB` into memory. Memory address is formed by adding `offsetField` with the contents of `regA`. Behavior is defined only for memory addresses in the range [0, 65535].
`beq` (I-type instruction)	0b100	"Branch if equal" If the contents of `regA` and `regB` are the same, then branch to the address `PC+1+offsetField`, where `PC` is the address of this beq instruction.
`jalr` (J-type instruction)	0b101	"Jump and Link Register"; **First** store the value `PC+1` into `regB`, where `PC` is the address where this `jalr` instruction is defined. **Then** branch (set PC) to the address contained in `regA`. **Note** that this implies if `regA` and `regB` refer to the same register, the net effect will be jumping to `PC+1`.
`halt` (O-type instruction)	0b110	Increment the `PC` (as with all instructions), then halt the machine (let the simulator notice that the machine halted).
`noop` (O-type instruction)	0b111	"No Operation (pronounced no op)" Do nothing.



```c
int main(int argc, char** argv) {
    instruction all_lines[MAXLINELENGTH]; // 已经读好
    int line_num = 0; // 为实际行数，已经保存好
    if (check_duplicate_label(all_lines, line_num)) {
        my_exit(all_lines, line_num); }
    char text_data[MAXLINELENGTH * 12]; text_data[0] = '\0';
    char symbol[MAXLINELENGTH * 12]; symbol[0] = '\0';
    char relocation[MAXLINELENGTH * 12]; relocation[0] = '\0';
    int text_num = 0; int data_num = 0; int symbol_num = 0;
    int reloc_num = 0; int fill_start = 0; bool _fill = false;
    char all[MAXLINELENGTH * 12]; all[0] = '\0';
    for (int i = 0; i < line_num; i++) {
        if (strlen(all_lines[i].label) > 0 && is_upper(all_lines[i].label[0])) {
            symbol_num++;
            if (strcmp(all_lines[i].opcode, ".fill") == 0) {
                sprintf(symbol + strlen(symbol), "%s D %d\n", all_lines[i].label, (_fill ? i - fill_start : 0)); } else
                sprintf(symbol + strlen(symbol), "%s T %d\n", all_lines[i].label, i); } }
        if (strcmp(all_lines[i].opcode, "add") == 0) {
            text_num++; instruction current = all_lines[i];
            if (isNumber(current.field1) == 0 || isNumber(current.field2) == 0 || isNumber(current.field3) == 0) {
                my_exit(all_lines, line_num); }
            int f1 = atoi(current.field1); int f2 = atoi(current.field2); int f3 = atoi(current.field3);
            if (register_invalid(f1) || register_invalid(f2) || register_invalid(f3)) {
                my_exit(all_lines, line_num); }
            int result = 0; result = f3 + (f2 << 16) + (f1 << 19) + (0 << 22);
            sprintf(text_data + strlen(text_data), "%d\n", result);
        } else if (strcmp(all_lines[i].opcode, "nor") == 0) {
            text_num++; instruction current = all_lines[i];
            if (isNumber(current.field1) == 0 || isNumber(current.field2) == 0 || isNumber(current.field3) == 0) {
                my_exit(all_lines, line_num); }
            int f1 = atoi(current.field1); int f2 = atoi(current.field2); int f3 = atoi(current.field3);
            if (register_invalid(f1) || register_invalid(f2) || register_invalid(f3)) {
                my_exit(all_lines, line_num); }
            int result = 0; result = f3 + (f2 << 16) + (f1 << 19) + (1 << 22);
            sprintf(text_data + strlen(text_data), "%d\n", result);
        } else if (strcmp(all_lines[i].opcode, "lw") == 0) {
            text_num++; instruction current = all_lines[i];
            if (isNumber(current.field1) == 0 || isNumber(current.field2) == 0) {
                my_exit(all_lines, line_num); }
            int f1 = atoi(current.field1); int f2 = atoi(current.field2);
            if (register_invalid(f1) || register_invalid(f2)) {
                my_exit(all_lines, line_num); }
            int f3 = -2;
            if (isNumber(current.field3)) {
                f3 = atoi(current.field3);
                if (f3 < -32768 || f3 > 32767) {
                    my_exit(all_lines, line_num); }
            } else {
                f3 = find_label(current.field3, all_lines, line_num);
                if (f3 == -1) {
                    if (is_lower(current.field3[0])) {
                        my_exit(all_lines, line_num); }
                    f3 = 0;
                    if (strstr(symbol, current.field3) == NULL) {  // 还没有记录
                        symbol_num++; sprintf(symbol + strlen(symbol), "%s U %d\n", current.field3, 0); } }
                reloc_num++; sprintf(relocation + strlen(relocation), "%d lw %s\n", i, current.field3);
            }
            f3 = f3 & ((1 << 16) - 1); int result = 0;
            result = f3 + (f2 << 16) + (f1 << 19) + (2 << 22); sprintf(text_data + strlen(text_data), "%d\n", result);
        } else if (strcmp(all_lines[i].opcode, "sw") == 0) {
            text_num++; instruction current = all_lines[i];
            if (isNumber(current.field1) == 0 || isNumber(current.field2) == 0) {
                my_exit(all_lines, line_num); }
            int f1 = atoi(current.field1); int f2 = atoi(current.field2);
            if (register_invalid(f1) || register_invalid(f2)) {
                my_exit(all_lines, line_num); }
            int f3 = -2;
            if (isNumber(current.field3)) {
                f3 = atoi(current.field3);
                if (f3 < -32768 || f3 > 32767) {
                    my_exit(all_lines, line_num); }
            } else {
                f3 = find_label(current.field3, all_lines, line_num);
                if (f3 == -1) {
                    if (is_lower(current.field3[0])) {
                        my_exit(all_lines, line_num); }
                    f3 = 0;
                    if (strstr(symbol, current.field3) == NULL) {  // 还没有记录
                        symbol_num++; sprintf(symbol + strlen(symbol), "%s U %d\n", current.field3, 0); }
                }
                reloc_num++; sprintf(relocation + strlen(relocation), "%d sw %s\n", i, current.field3);
            }
            f3 = f3 & ((1 << 16) - 1); int result = 0;
            result = f3 + (f2 << 16) + (f1 << 19) + (3 << 22); sprintf(text_data + strlen(text_data), "%d\n", result);
        } else if (strcmp(all_lines[i].opcode, "beq") == 0) {
            text_num++; instruction current = all_lines[i];
            if (isNumber(current.field1) == 0 || isNumber(current.field2) == 0) {
                my_exit(all_lines, line_num);
            }
            int f1 = atoi(current.field1); int f2 = atoi(current.field2);
            if (register_invalid(f1) || register_invalid(f2)) {
                my_exit(all_lines, line_num); }
            int f3 = -2;
            if (isNumber(current.field3)) {
                f3 = atoi(current.field3);
                if (f3 < -32768 || f3 > 32767) {
                    my_exit(all_lines, line_num); }
            } else {
                int target = find_label(current.field3, all_lines, line_num);
                if (target == -1) {
                    my_exit(all_lines, line_num); }
                f3 = target - (i + 1);
            }
            f3 = f3 & ((1 << 16) - 1); int result = 0;
            result = f3 + (f2 << 16) + (f1 << 19) + (4 << 22); sprintf(text_data + strlen(text_data), "%d\n", result);
        } else if (strcmp(all_lines[i].opcode, "jalr") == 0) {
            text_num++; instruction current = all_lines[i];
            if (isNumber(current.field1) == 0 || isNumber(current.field2) == 0) {
                my_exit(all_lines, line_num); }
            int f1 = atoi(current.field1); int f2 = atoi(current.field2);
            if (register_invalid(f1) || register_invalid(f2)) {
                my_exit(all_lines, line_num); }
            int result = 0; result = (f2 << 16) + (f1 << 19) + (5 << 22);
            sprintf(text_data + strlen(text_data), "%d\n", result);
        } else if (strcmp(all_lines[i].opcode, "noop") == 0) {
            text_num++; int result = 0;
            result = (7 << 22); sprintf(text_data + strlen(text_data), "%d\n", result);
        } else if (strcmp(all_lines[i].opcode, "halt") == 0) {
            text_num++; int result = 0;
            result = (6 << 22); sprintf(text_data + strlen(text_data), "%d\n", result);
        } else if (strcmp(all_lines[i].opcode, ".fill") == 0) {
            if (!_fill) {
                fill_start = i;  _fill = true; }
            data_num++; instruction current = all_lines[i]; int f3 = -2;
            if (isNumber(current.field1)) {
                f3 = atoll(current.field1);
                if (f3 < -2147483648 || f3 > 2147483647) {
                    my_exit(all_lines, line_num); }
            } else {
                f3 = find_label(current.field1, all_lines, line_num);
                if (f3 == -1) {
                    if (is_lower(current.field1[0])) {
                        my_exit(all_lines, line_num); }
                    f3 = 0;
                    if (strstr(symbol, current.field1) == NULL) {  // 还没有记录
                        symbol_num++;
                        sprintf(symbol + strlen(symbol), "%s U %d\n", current.field1, 0); } }
                reloc_num++; sprintf(relocation + strlen(relocation), "%d .fill %s\n", i - fill_start, current.field1);
            }
            int result = 0; result = f3; sprintf(text_data + strlen(text_data), "%d\n", result);
        } else {
            my_exit(all_lines, line_num); }
    }
    sprintf(all + strlen(all), "%d %d %d %d\n", text_num, data_num, symbol_num, reloc_num);
    sprintf(all + strlen(all), "%s%s%s", text_data, symbol, relocation);
}
```