گزارش پیاده سازی تکلیف 5 هم طراحی سخت افزار و نرم افزار

غزل کلهری 97243058

عرفان مشیری 97243066

هدف از انجام این تمرین پیاده سازی معماری Micro-Programmed بود. بدین ترتیب که بر روی ماشین پیاده سازی شده در کتاب، طبق خواسته های صورت سوال تغییراتی ایجاد کردیم.

1. ابتدا مجموعه دستورات داده شده را در دو بخش define مرتبط با ALU encoding و ماژول alu اضافه کردیم. (خطوط هایلایت شده تغییرات ایجاد شده در هر بخش را نشان میدهند.)

```
/* encoding for ALU */
#define ALU_ACC 0 /* ALU <- ACC */
#define ALU_PASS 1 /* ALU <- SBUS */
#define ALU_ADD 2 /* ALU <- ACC + SBUS */
#define ALU_SUBA 3 /* ALU <- ACC - SBUS */
#define ALU_SUBS 4 /* ALU <- SBUS - ACC */
#define ALU_AND 5 /* ALU <- ACC and SBUS */
#define ALU_OR 6 /* ALU <- ACC or SBUS */
#define ALU_NOT 7 /* ALU <- not SBUS */
#define ALU_INCS 8 /* ALU <- SBUS + 1 */
#define ALU_INCA 9 /* ALU <- ACC + 1 */
#define ALU_CLR 10 /* ALU <- 0 */
#define ALU_SET 11 /* ALU <- 1 */
#define ALU_X ALU_ACC /* don't care */
#define ALU_ADDI 12 /* ALU <- ACC + Num */
#define ALU_SUBIA 13 /* ALU <- Num - ACC */
#define ALU_SUBAI 14 /* ALU <- ACC - Num */
```

```
dp alu (in ctl_alu : ns(4);
        in sbus : ns(WLEN);
        in acc : ns(WLEN);
        in num : ns(4);      // ADDED TO USE AS BIT NUMBER 27-31
        out q : ns(WLEN)) {
    always {
        q = (ctl_alu == ALU_ACC) ? acc :
            (ctl_alu == ALU_PASS) ? sbus :
            (ctl_alu == ALU_ADD) ? acc + sbus :
            (ctl_alu == ALU_SUBA) ? acc - sbus :
            (ctl_alu == ALU_SUBS) ? sbus - acc :
            (ctl_alu == ALU_AND) ? acc & sbus :
            (ctl_alu == ALU_OR) ? acc | sbus :
            (ctl_alu == ALU_NOT) ? ~sbus :
            (ctl_alu == ALU_INCS) ? sbus + 1 :
            (ctl_alu == ALU_INCA) ? acc + 1 :
            (ctl_alu == ALU_CLR) ? 0 :
            (ctl_alu == ALU_SET) ? 1 :
            (ctl_alu == ALU_ADDI) ? acc + num :
            (ctl_alu == ALU_SUBAI) ? acc - num :
            (ctl_alu == ALU_SUBIA) ? num - acc :
            0;
    }
}
```

2. در ادامه واحد acc ساختار alu را از آن جدا کرده و بصورت ماژولی جداگانه تعریف کردیم.

```
dp acc (in ctl_dest : ns(4);
        in shift_out : ns(WLEN);
        out q : ns(WLEN)) {
    reg acc : ns(WLEN);
    always {
        acc = (ctl_dest == DST_ACC) ? shift_out : acc;
        q = acc;
    }
}
```

3. خط مربوط به مالتی پلکسر register file, input را حذف کرده و بجای آن ماژولی جداگانه تعریف کردیم.

```
//sbus = (ctl_sbus == SBUS_IN) ? din : rf_out;
```

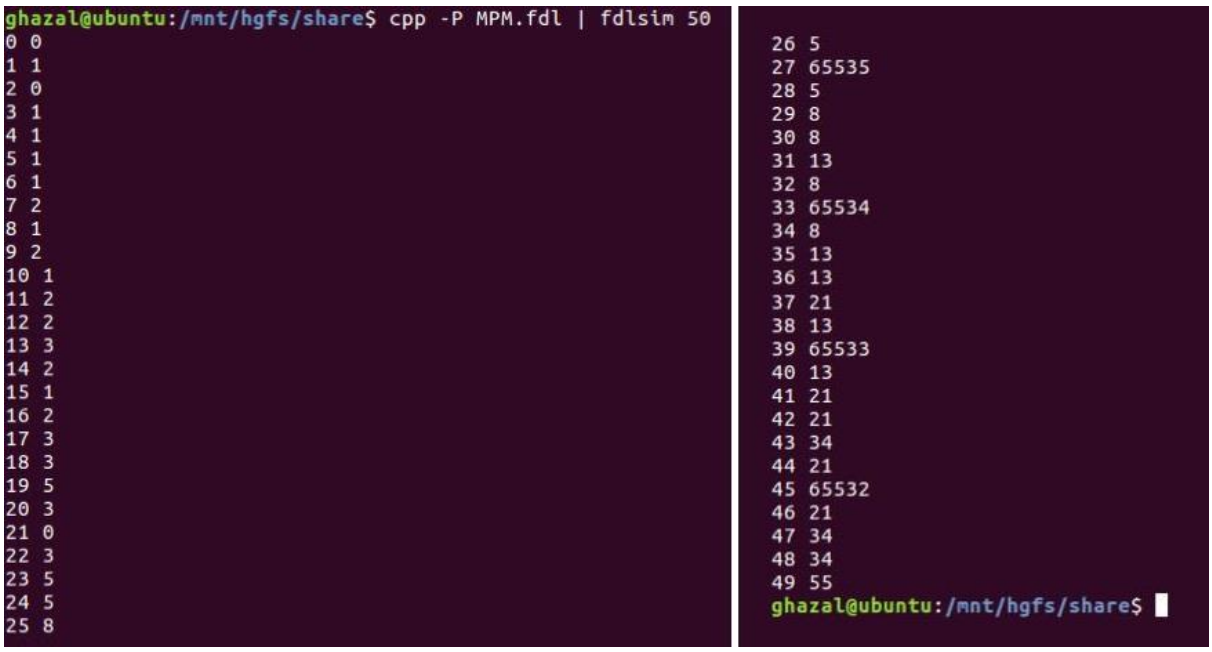
```
dp mux_rf_in(in din : ns(WLEN);
             in ctl_sbus : ns(4);
             in rf_out :ns(WLEN);
             out dout : ns(WLEN) ) {
    always {
        dout = (ctl_sbus == SBUS_IN) ? din : rf_out;
    }
}
```

4. با تغییر MI های موجود در lookup، برنامه تولید کننده اعداد فیبوناچی را با برنامه قبلی جایگزین کردیم.

```
lookup cstore : ns(32) = {
    // 0: 1 -> R3 (counter)
    MI(O_NIL, SBUS_X, ALU_SET, SHFT_NIL, DST_R3, NXT_NXT, 0),
    // 1: R3 << 3 -> R3 (R3 = 8) (we have first and second number of sequence)
    MI(O_NIL, SBUS_R3, ALU_PASS, SHFT_SHL, DST_R3, NXT_NXT, 0),
    // 2: 1 -> R0 (first number)
    MI(O_NIL, SBUS_X, ALU_SET, SHFT_NIL, DST_R0, NXT_NXT, 0),
    // 3: 1 -> R1 (second number)
    MI(O_NIL, SBUS_X, ALU_SET, SHFT_NIL, DST_R1, NXT_NXT, 0),
    // 4: R0 -> ACC (we use ACC to add numbers)
    MI(O_NIL, SBUS_R0, ALU_PASS, SHFT_NIL, DST_ACC, NXT_NXT, 0),
    // 5: R1 + ACC -> R2 (result)
    MI(O_NIL, SBUS_R1, ALU_ADD, SHFT_NIL, DST_R2, NXT_NXT, 0),
    // 6: R1 -> R0 (next num)
    MI(O_NIL, SBUS_R1, ALU_PASS, SHFT_NIL, DST_R0, NXT_NXT, 0),
    // 7: R2 -> R1 (next num)
    MI(O_NIL, SBUS_R2, ALU_PASS, SHFT_NIL, DST_R1, NXT_NXT, 0),
    // 8: 1 -> ACC
    MI(O_NIL, SBUS_X, ALU_SET, SHFT_NIL, DST_ACC, NXT_NXT, 0),
    // 9: R3 - ACC -> R3 (check if 10 numbers are reched)
    MI(O_NIL, SBUS_R3, ALU_SUBS, SHFT_NIL, DST_R3, NXT_JMP, 4),
    // 10 Ldone: R2 -> OUT || JUMP Lstart
    MI(O_WR, SBUS_R2, ALU_X, SHFT_X, DST_X, NXT_JMP, 10)
};
```

با توجه به کامنت های نوشته شده، دو جمله اول دنباله را در رجیسترهای 0 و 1 ذخیره کرده، از ACC به عنوان متغیر کمکی استفاده میکنیم تا هربار این دو عدد را با یکدیگر جمع کنیم. سپس نتیجه را در رجیستر 2 ریخته و رجیسترهای 0 و 1 را با مقادیر جدید آپدیت میکنیم. و به ابتدای حلقه تشکیل شده جامپ میزنیم تا عملیات دوباره تکرار شود (شبیه سازی بازگشتی).

5. در نهایت برنامه را با شبیه ساز GEZEL اجرا کردیم. نتایج حاصل از اجرای برنامه در ادامه آورده شده است. خروجی برنامه در طی سایکل های زمانی طی شده، اعداد تشکیل دهنده دنباله فیبوناچی را به ما نشان میدهد.



- با توجه به بخش های اضافه شده در مراحل قبل، تعداد کمی ویرایش در dp hmm داریم که همگی با کامنت در کد ضمیمه شده مشخص شده اند.